Configuring

Table of Contents

You must fine-tune Memory and Database settings for better performance before using Restcomm SMSC in production. Once you complete setting up the Gateway you must configure the SS7 Stack the SMSC Gateway following the instructions specified in this guide.

Memory Settings

You should fine tune the JVM memory settings based on your needs but we recommend you allocate a minimum of 3 GB for initial and maximum heap size. These settings are specified in the file *restcomm-smscgateway-version>/jboss-5.1.0.GA/bin/run.conf*.

-Xms3072m

Initial heap size, set in megabytes

-Xmx3072m

Maximum heap size, set in megabytes

Configuring log4j Logging Service

Restcomm SMSC uses `Apache log4j` for logging. If you are not familiar with the `log4j` package, you can read more about it at the Jakarta website.

Logging is controlled from a central configuration file located at *restcomm-smscgateway-<version>/jboss-5.1.0.GA/server/<profile>/conf/jboss-log4j.xml*, one for each JBoss AS configuration profile. This file defines a set of appenders specifying the log files, what categories of messages should go there, the message format and the level of filtering. For more details, please refer to Section 9.6.3, "Logging Service" in the JBoss AS Getting Started Guide available here.

You must make sure `log4j` is fine tuned for optimal performance in production. We recommend that you set logging threshold to `WARN` and let the CDR appender be `DEBUG`.

SMSC GW routing fundamentals

SMSC GW needs clear info to where it must route messages that have come to SMSC GW from SMPP / SS7 / SIP connectors or created inside SMSC GW receipts. Routing procedures gives the answer to this questions, that a user must configure before SMSC GW can properly work.

SMSC GW has two routing procedures:

- The procedure based on dividing of separated routing area into several areas ("subnetworks") and routing of messages between subnetworks.
- The procedure of routing messages to destinations inside subnetworks.

The procedure based on dividing of separated routing area into several areas ("subnetworks").

Each area has its unique digital "netwokId" value. Each SMPP connection (ESME), SS7 SCCP service access point (SAP), SIP connector and a database routing rule (if you use them instead of default ESME routing rules) belongs to one of network (to each of the a networkId value is assigned). Default netwokId value is "0". If a user does not specify a netwokId value when ESME, SAP, SIP or a database routing rule creation, networkId=0 will be assigned.

When a message has come to SMSC GW a networkId value from ESME, SAP or SIP is assigned for the message. The message can be routed only inside the area with the same networkId to what the message belongs. So this means that by default a message can not leave a "networkId" area via which it has come to SMSC GW.

If a user needs that a message will be delivered to another "networkId" area, he can configure "message processing" rules ("mproc" rules). There we can configure that a message from a specified networkId1 and which fits some other conditions (like destination address fit to some mask) must be moved to networkId2 (newNetworkId option of a mproc rule). And the message will be delivered via ESME, SAP or SIP that belongs to networkId2.

Multi-tenancy support is also based on this routing procedure. For each networkId we can configure a separate SS7 GlobalTitle and for an external peer each networkId can play a role as a separate independed SMSC.

NetworkId for a generated inside SMSC receipt is assigned by two algoritms:

- if the option "Delivery receipts will be routed to the origination networkId" is set to true - networkId from ESME/SIP/SAP via what an original message has left SMSC GW will be assigned
- if the option "Delivery receipts will be routed to the origination networkId" is set to false - networkId from ESME/SIP/SAP via what an original message has come to SMSC GW will be assigned

NetworkId are specified at different places:

- for JSS7 level SAPs and SCCP routing rules are configured for this. See "7.4.5. Create a new Service Access Point" and "7.4.17. Create a new SCCP Rule" chapters of "JSS7 stack User Guide".
- for ESMEs see the chapter [_esme_settings].
- for SIP level see the chapter [_sip].
- for database routing rules see [_db_routing_rule_settings]

For configuring of mproc rules - see chapters Message processing rules (mproc rules) and [_mproc_rule_settings].

For configuring of option "Delivery receipts will be routed to the origination networkId" - see the chapter [_smsc_gateway_server_routing_dlv].

For configuring of SMSC Global Titles - see chapters [_set_scgt].

The procedure of routing messages to destinations inside subnetworks.

Inside each subnetwork there may be several ESME, SAP or SIP configured. If we use default routing rules, then for routing SMSC GW makes following steps:

- it checks all ESMEs that belongs to the networkId. It checks if TON, NPI and AddressRange of an ESME Routing Address fit to a message destination TON, NPI and address digits the the message will be routed to this ESME. There is one exception of this rule. A message will not be routed in any case to the ESME from which it has come.
- if no ESME found for routing SMSC GW then checks SIP connector if it's settings (TON, NPI and routing AddressRange) fits to a message destination TON, NPI and address digits. If yes the message will be routed to SIP
- if SIP also does not fit, a message will be routed to SS7 network. Then routing rules at SCCP level will be taken into account for further routing.

For configuring of ESME and SIP routing parameters see [_esme_settings] and [_sip].

Instead of default routing rules database routing rules can be used. This is a set of stored in the cassandra database rules that describes where SMSC GW will route a message (to some ESME / SIP) depending on a message destination address. Messages that fit to no rules will be routed to SS7 network. For configuring of database routing rules see the chapter [_mproc_rule_settings]| |

Using of different routing procedures.

You can use only the procedure based on subnetworks (networkId) areas, only the procedure of routing messages inside subnetworks or both.

If you want to use only the procedure based on subnetworks (networkId) areas you need:

- configure SMSC GW so that only one SAP / ESME / SIP belongs to each networkId.
- for every ESME / SIP specify for "Routing Type of number (TON)"=-1, "Routing Number plan indicator (NPI)"=-1, "Routing Range"="^[0-9a-zA-Z]*" (that fits to any message destination address).
- configure a set of mproc rules that will manage of routing of messages from one subnetwork (networkId) to another.| |

If you want to use only the procedure based on subnetworks (networkId) areas you need:

- configured for all SAP / SCCP routing rules / ESME / SIP the same networkId (default networkId=0 is usually used)
- for every ESME / SIP specify proper values for "Routing Type of number (TON)", "Routing Number plan indicator (NPI)" and "Routing Range" or configure database routing rules
- you do not need to configure mproc rules for routing| |

If you want to use both, you need to configure both parts. Messages will be routed between

subnetworks (networkIds) by mproc rules and inside a subnetwork (networkId) by configuring of ESME / SIP "Routing Type of number (TON)", "Routing Number plan indicator (NPI)" and "Routing Range" (or database routing rules).

Routing model examples.

If you have one ESME and JSS7 connector, and the only traffic is ESME → JSS7

- use networkId=0 for all
- configure "Routing Range" value of the ESME to a value that fits to no mobile subscriber (for example "0")

If you use the model "all JSS7 originated messages must be routed to ESME and all ESME originated messages must be routed to JSS7", you can:

- set networkId=0 for JSS7 and networkId=1 for ESME
- set ESME routing address so it accepts all messages
- set mproc rules:

  smsc mproc add mproc 1 networkIdMask=1 newNetworkId=0

  smsc mproc add mproc 2 networkIdMask=0 newNetworkId=1

If you have several ESMEs that send message to one SS7 connection and you need that delivery receipts come back to an originator ESME (and no more traffic), you can:

- put SS7 SAP / SCCP rules to networkId=0
- put each ESME to each own networkId (1,2,3)
- create a set of mproc rules that move ESME originated messages into SS7 network. CLI commabd can be like:

  smsc mproc add mproc 1 networkIdMask=1 newNetworkId=0

  smsc mproc add mproc 2 networkIdMask=2 newNetworkId=0

  smsc mproc add mproc 3 networkIdMask=3 newNetworkId=0

- set the option "Delivery receipts will be routed to the origination networkId" is set to true

Message processing rules (mproc rules)

Message processing rules (mproc rules) is a tool for processing messages and changing properties of message, for example source/destination TON, NPI, NetworkId etc. MProc rules are only applied if pre configured criterian match’s.

Following are the states at which mproc rules can be applied to messages:

`onPostArrival` : When a message arrives in SMSC GW and has been already processed (accepted) by Diameter server (if Diameter server is configured). `onPostArrival` following actions are possible:

- message can be dropped (a success response will be returned to a message originator)
- message can be rejected (a reject response will be returned to a message originator)
- most of a message parameters can be updated (for example destination address, networkId (this is needed for routing such messages to another subnetwork area (networkId) (see chapter SMSC GW routing fundamentals)) or even a message content)
- a new message(es) can be posted for delivering. To post new messages at this step no Diameter server request and no mproc rules will be applied

Only after actions applying messages will be stored into a database or/and delivered to a destination.

- `onPostImsiRequest` : When a successful SRI response has been received from HLR for an SS7 destination message. At this step a message can be dropped (to prevent further delivery). This can be very useful feature to prevent SMS being delivered to off-net or roaming subscribers.
- `onPostDelivery` : When a message delivery was succeeded or failed. At this step a new message(es) can be posted for delivering (it can be for example some delivery report). | |

SMSC comes with predifined set of mproc rules (default implementation) (see chapter Default message processing rules implementation). However users can make their own customized implementation of mproc rules by using java programming and implementing provided interfaces (see chapter Customized message processing rules). All mproc rules implementation has its Class Name. The Class Name of a default implementation is "mproc". This is the name by which users can create new mproc rule instances.

Users can create one or more mproc rules, modify, show and remove some of them by CLI or GUI interface. See details in chapter [_mproc_rule_settings]. Each mproc rule has it's unique serial id. Mproc rule are sorted by this id value. SMSC appliec mproc rules to a message in ascending order, that is mproc rule with the least id is applied before mproc rule with next id etc.

While checking if mproc rule conditions match to a message, updates (that were made after previous rules applying) are taken into account. For example if a message destination address has been changed by rule 1, then rule 2 will check if this updated destination address matches to rules 2 or not.

Default message processing rules implementation

SMSC GW contains a default implementation of mproc rules that cover some requirenments. Information how to manage rules can be found in chapter [_mproc_rule_settings]. This chapter covers a description of conditions and actions that are present in default mproc rules implementation.

Parameters for mproc rule are divided into two cathegories:

1. a) Conditions. If a message fits to all conditions then the rule will be applied to the message.

Table 1. The list of possible conditions

Parameter name	Value	Description	Default value
desttonmask	<destination type of number>	mproc rule will be applied only if message destination Type of Number is equal to this value "desttonmask"	"-1" : acts as wild card and hence messages with any TON will match.
destnpimask	<destination numbering plan indicator>	mproc rule will be applied only if message destination Numbering Plan Indicator is equal to this value "destnpimask".	"-1" : acts as wild card and hence messages with any NPI will match.
destdigmask	<java regular expression - destination number digits mask>	mproc rule will be applied only if message destination address digits match's with "destdigmask" java regular expression.	"-1" : acts as wild card and hence messages with any destination number will match.
originatingmask	<SS7_MO | SS7_HR | SMPP | SIP>	mproc rule will be applied only if message arrived in SMSC GW via a defined "originatingmask" connector. . SS7_MO: SMS Originated from Mobile - SS7 connection . SS7_HR: SMS Originated from Home Routing configuartion - SS7 connection . SMPP: SMS Originated from SMPP connector and SIP connector. . SIP: SMS Originated from SIP connector.	"-1" : acts as wild card and hence messages originated from any channel will match.
originatorsccpaddress-mask	<java regular expression - originator CallingPartyAddress digits mask>	mproc rule will be applied only if CallingPartyAddress digits match's with "originatorsccpaddress mask" java regular expression.	"-1" : acts as wild card and hence messages with any CallingPartyAddress digits or without it will match.

Parameter name	Value	Description	Default value
networkidmask	<networkId value>	mproc rule will be applied only if message has come to SMSC GW via a subnetwork with networkId which is equal to this value "networkidmask".	"-1" : acts as wild card and hence messages from any network will match.
origesmenamemask	<java regualar expression - origination ESME name mask>	mproc rule will be applied only if message has come to SMSC GW from SMPP connector from ESME with a name that fits "origesmenamemask".	"-1" : acts as wild card and hence any message will match (never mind if it came from SS7 or SIP).

b) Actions, which will be applied to messages.

Table 2. The list of possible actions

Parameter name	Value	Description	Default value
newnetworkid	<new networkId value>	networkId of the message will be changed to "newnetworkid" value. This means that the message will be delivered via connectors that belong to the new networId.	"-1". This means that networkId of the message will not be changed.
newdestton	<new destination type of number>	a message destination Type of Number will be changed to "newdestton" value.	"-1". This means that destination Type of Number of the message will not be changed.
newdestnpi	<new destination numbering plan indicator>	a message destination Numbering Plan Indicator will be changed to "newdestnpi" value.	"-1". This means that destination Numbering Plan Indicator of the message will not be changed.

Parameter name	Value	Description	Default value
adddestdigprefix	<prefix>	adddestdigprefix will be added into a begin of a message destination address digits. For example if adddestdigprefix is "22" and destination address digits are "3333333", then the new value of destination address digits will be "223333333".	"-1". This means that destination address digits of the message will not be changed.
makecopy	<false | true>	If the makecopy action is present then SMSC GW makes a copy of a message and then post the copy for further message processing in addition to the original message. All other actions in the rule will be applied only to the copy. For example user wants to make copies of messages and send them to another destinations (by sending of copies into another networkId), then for this you can create a rule and set for the rule "makecopy true" and "newnetworkid <new networkId value>" parameters. This makes a copy of a message and set for the copy a new networkId value.	false

Parameter name	Value	Description	Default value
dropaftersri	<false | true>	This is an action at the step when a successful SRI response has been received from HLR for an SS7 destination message. If a SRI success response has received then the message will be dropped without delivery attempt. A delivery response in this case will contain extra fields (IMSI and NetworkNodeNumber values).	false

Customized message processing rules

Default mproc rules allows to change the properties of a message in pre defined way, however if user wants to achieve more, SMSC allows users to implement their own custom logic. Below steps describes how to implement custom mproc rules.

Procedure: Steps for custom mproc rules implementing

1. User should implement the business logic as java code. There are couple of interfaces exposed by SMSC, `MProcRuleFactory` and `MProcRule` that must be implemented and add the custom business logic. User needs to cover two parts of rules usage - a rule configuring part and a rule applying part.

   

   Please pay attention that in your code you may not perform long delays in order not to dramatically decrease of SMSC GW productivity.

   Once custom rule is implemented, user will have to create jar file and deploy it into SMSC Gateway.

   - User needs to decide a rule class name that will be used to uniquely identify the custom rules. This can be any word without spaces. For default mproc rules implementation the rule class name is "mproc". For example consider rule class name as "testrule".
   - User needs to impelent two interfaces: `MProcRuleFactory` and `MProcRule`. For example consider `MProcRuleFactoryTestImpl` and `MProcRuleTestImpl`.
   - User needs to decide which actions will custom rules perform and for which messages. For example create a custom rule that will be applied to any message at `onPostArrival` state who's destination address digits are starting with a configurable parameter "par1". For all the messages which match's this rules condition, prefix "par2" (the configurable parameter) is to be applied to message destination address.

2. Creating custom classes:

```
package org.mobicents.smsc.mproc.testimpl;

import org.mobicents.smsc.mproc.MProcRuleFactory;

public class MProcRuleFactoryTestImpl implements MProcRuleFactory {
}
```

```
package org.mobicents.smsc.mproc.testimpl;

import org.mobicents.smsc.mproc.MProcRule;

public class MProcRuleTestImpl implements MProcRule {
}
```

3. Implementing `MProcRuleFactory` interface. The interface is:

```
package org.mobicents.smsc.mproc;

public interface MProcRuleFactory {
    String getRuleClassName();
    MProcRule createMProcRuleInstance();
}
```

Method `getRuleClassName()` must return the rule class name. Method `createMProcRuleInstance()` must return a custom implementation of `MProcRule` (in this example instance of `MProcRuleTestImpl` class). Here is an example of implementation:

```
package org.mobicents.smsc.mproc.testimpl;

import org.mobicents.smsc.mproc.MProcRule;
import org.mobicents.smsc.mproc.MProcRuleFactory;

public class MProcRuleFactoryTestImpl implements MProcRuleFactory {
    public static final String CLASS_NAME = "testrule";

    @Override
    public String getRuleClassName() {
        return CLASS_NAME;
    }

    @Override
    public MProcRule createMProcRuleInstance() {
        return new MProcRuleTestImpl();
    }

}
```

4. Next is actual business logic that should go in implement of `MProcRule` interface. Let's start with learning of the interface. The content of the interface is the following:

```
package org.mobicents.smsc.mproc;

public interface MProcRule extends MProcRuleMBean {

    void setId(int val);

    /**
     * @return true if the mproc rule fits to a message when a message has just
come to SMSC
     */
    boolean matchesPostArrival(MProcMessage message);

    /**
     * @return true if the mproc rule fits to a message when IMSI / NNN has been
received from HLR
     */
    boolean matchesPostImsiRequest(MProcMessage message);

    /**
     * @return true if the mproc rule fits to a message when a message has just
been delivered
     * (or delivery failure)
     */
    boolean matchesPostDelivery(MProcMessage message);

    /**
```

```
     * the event occurs when a message has just come to SMSC
     */
    void onPostArrival(PostArrivalProcessor factory, MProcMessage message) throws
Exception;

    /**
     * the event occurs when IMSI / NNN has been received from HLR
     */
    void onPostImsiRequest(PostImsiProcessor factory, MProcMessage message) throws
Exception;

    /**
     * the event occurs when a message has just been delivered (or delivery
failure)
     */
    void onPostDelivery(PostDeliveryProcessor factory, MProcMessage message) throws
Exception;

    /**
     * this method must implement setting of rule parameters as for provided CLI
string at
     * the step of rule creation
     */
    void setInitialRuleParameters(String parametersString) throws Exception;

    /**
     * this method must implement setting of rule parameters as for provided CLI
string at
     * the step of rules modifying
     */
    void updateRuleParameters(String parametersString) throws Exception;

}
```

and the parent interface content is:

```
package org.mobicents.smsc.mproc;

public interface MProcRuleMBean {

    /**
     * @return the id of the mproc rule
     */
    int getId();

    /**
     * @return Rule class of the mproc rule ("default" or other when a customer
implementation)
     */
    String getRuleClassName();

    /**
     * @return true if the mproc rule is used for the phase when a message has just
come to SMSC
     */
    boolean isForPostArrivalState();

    /**
     * @return true if the mproc rule is used for the phase when IMSI / NNN has
been received
     * from HLR
     */
    boolean isForPostImsiRequestState();

    /**
     * @return true if the mproc rule is used for the phase when a message has just
been
     * delivered (or delivery failure)
     */
    boolean isForPostDeliveryState();

    /**
     * @return rule parameters as CLI return string
     */
    String getRuleParameters();

}
```

Table 3. `MProcRule` and `MProcRuleMBean` interfaces methods description

Methods	Description
getId(), setId()	mproc rule id getter and setter. id value is unique for each mproc rule inside SMSC GW.
getRuleClassName()	Returns the rule class name ("testrule" in this example)

Methods	Description
getRuleParameters(), setInitialRuleParameters(), updateRuleParameters()	Getter and setters of configured parameters for a rule instance. Parameters are formed as a plain text string, that can be provided by CLI interface. In this example we need to configure two parameters par1 and par2. Let's specify the parameters strinf as "par1" and "par2" (two values after a space, for example "11 34").
isForPostArrivalState(), isForPostImsiRequestState(), isForPostDeliveryState()	Returns true if this rule affects to a message processing step (for onPostArrival, onPostDelivery and onPostImsi requests) and false if not. You have to check configured parameters of a rule and return a proper value
matchesPostArrival(MProcMessage message), matchesPostImsiRequest(MProcMessage message), matchesPostDelivery(MProcMessage message)	These methods must return true if rule should be applied false if not (for onPostArrival, onPostImsi and onPostDelivery requests).
onPostArrival(), onPostImsiRequest(), onPostDelivery()	These methods are needed for implementing of rule applying actions.

Id parameter and custom rule parameters ("par1" and "par2" in this example) must be stored into xml config file (it is located in *restcomm-smscgateway-<version>/jboss-5.1.0.GA/server/<profile>/data/SmscManagement_mproc.xml*). For this user needs to implement several extra methods that will be described later.

5. Reusing of base `MProcRuleBaseImpl` class. It is recommended for your custom `MProcRuleTestImpl` to extend `MProcRuleBaseImpl` provided by SMSC. `MProcRuleBaseImpl` class contains some useful methods that cover getter and setter and code for persisting of this parameter into xml config file (read() and write() methods), default stubs for `matches()` and `onPost*()` methods and a service method for splitting of a parameters plain text string into subparameters (splitParametersString()).

   Here is the new template of `MProcRuleTestImpl` class implementation that reuses the base class `MProcRuleBaseImpl`:

   ```
   package org.mobicents.smsc.mproc.testimpl;

   import org.mobicents.smsc.mproc.MProcRuleBaseImpl;

   public class MProcRuleTestImpl extends MProcRuleBaseImpl {
   }
   ```

6. Implementing of `getRuleClassName()` method of `MProcRule`. Below is an example:

```
    @Override
    public String getRuleClassName() {
        return MProcRuleFactoryTestImpl.CLASS_NAME;
    }
```

7. Implementing of methods that cover getter / setters for message custom parameters and storing them into xml config file.

   In the example setInitialRuleParameters() and updateRuleParameters() methods are implemented in the same way. You can implement them in different ways so that updateRuleParameters() method can accept not the whole set of parameters but only that ones that a user wants to change.

   M_PROC_RULE_TEST_XML is responsible for XML serializing / deserializing. See more info for them in javolution library specification.

   Here is an example (only the part that is described in this step):

```
package org.mobicents.smsc.mproc.testimpl;

import javolution.xml.XMLFormat;
import javolution.xml.stream.XMLStreamException;
import org.mobicents.smsc.mproc.MProcRuleBaseImpl;

public class MProcRuleTestImpl extends MProcRuleBaseImpl {

    private static final String PAR1 = "par1";
    private static final String PAR2 = "par2";
    private String par1, par2;

    @Override
    public void setInitialRuleParameters(String parametersString) throws Exception
{
        String[] args = splitParametersString(parametersString);
        if (args.length != 2) {
            throw new Exception("parametersString must contains 2 parameters");
        }
        par1 = args[0];
        par2 = args[1];
    }

    @Override
    public void updateRuleParameters(String parametersString) throws Exception {
        String[] args = splitParametersString(parametersString);
        if (args.length != 2) {
            throw new Exception("parametersString must contains 2 parameters");
        }
        par1 = args[0];
        par2 = args[1];
```

```
    }

    @Override
    public String getRuleParameters() {
        return par1 + " " + par2;
    }

    /**
     * XML Serialization/Deserialization
     */
    protected static final XMLFormat<MProcRuleTestImpl> M_PROC_RULE_TEST_XML = new
            XMLFormat<MProcRuleTestImpl>(MProcRuleTestImpl.class) {

        @Override
        public void read(javolution.xml.XMLFormat.InputElement xml,
MProcRuleTestImpl
            mProcRule) throws XMLStreamException {
            M_PROC_RULE_BASE_XML.read(xml, mProcRule);

            mProcRule.par1 = xml.getAttribute(PAR1, "");
            mProcRule.par2 = xml.getAttribute(PAR2, "");
        }

        @Override
        public void write(MProcRuleTestImpl mProcRule, javolution.xml.XMLFormat
.OutputElement
            xml) throws XMLStreamException {
            M_PROC_RULE_BASE_XML.write(mProcRule, xml);

            xml.setAttribute(PAR1, mProcRule.par1);
            xml.setAttribute(PAR2, mProcRule.par2);
        }
    };
}
```

8. Overiding one of `isForPost*()` methods (for allowing of processing of a needed message processing step). In our example it is `isForPostArrivalState()`:

```
    @Override
    public boolean isForPostArrivalState() {
        return true;
    }
```

9. Implementing of a needed `matches***()` methods. In our example we process messages which destination address digits are started with a configurable parameter "par1".

```
    @Override
    public boolean matchesPostArrival(MProcMessage message) {
        if (message.getDestAddr().startsWith(par1))
            return true;
        else
            return false;
    }
```

For `matches***()` methods we will use interface `MProcMessage` interface which provides info for processing message fields. Here is a code of these interfaces.

```
package org.mobicents.smsc.mproc;

import java.util.Date;

public interface MProcMessage {

    // source address part
    int getSourceAddrTon();

    int getSourceAddrNpi();

    String getSourceAddr();

    // dest address part
    int getDestAddrTon();

    int getDestAddrNpi();

    String getDestAddr();

    // message content part
    String getShortMessageText();

    byte[] getShortMessageBin();

    // other options
    int getNetworkId();

    int getOrigNetworkId();

    String getOrigEsmeName();

    OrigType getOriginationType();

    Date getScheduleDeliveryTime();

    Date getValidityPeriod();
```

```
    int getDataCoding();

    int getNationalLanguageSingleShift();

    int getNationalLanguageLockingShift();

    int getEsmClass();

    int getPriority();

    int getRegisteredDelivery();

    String getOriginatorSccpAddress();

}
```

10. Implementing of methods that make some actions: In example above this is `onPostArrival()` and the action is adding "par2" prefix into destination address digits. These methods will be invoked for all messages that match rule's conditions.

```
    @Override
    public void onPostArrival(PostArrivalProcessor factory, MProcMessage message)
throws Exception {
        String destAddr = this.par2 + message.getDestAddr();
        factory.updateMessageDestAddr(message, destAddr);
    }
```

In `onPostArrival()`, `onPostImsiRequest()` and `onPostDelivery()` methods user can use provided interfaces `PostArrivalProcessor`, `PostImsiProcessor` and `PostDeliveryProcessor` that provide methods for message processing / adding / dropping / rejecting.

- `PostArrivalProcessor` interface content:

```
package org.mobicents.smsc.mproc;

import java.util.Date;

import org.apache.log4j.Logger;

public interface PostArrivalProcessor {

    // access to environmental parameters
    /**
     * @return the logger that an application can use for logging info into
server.log
     */
    Logger getLogger();

    // actions
```

```
    /**
     * Drop the message. Success response (that a message is accepted) will be
return to
     * a message originator.
     */
    void dropMessage();

    /**
     * Drop the message. A reject will be sent to a message originator.
     */
    void rejectMessage();

    // updating of a message section
    void updateMessageNetworkId(MProcMessage message, int newNetworkId);

    /**
     * Updating of destination address message TON. In case of bad value (<0 or
>6)
     * MProcRuleException will be thrown
     *
     * @param message
     * @param newDestTon
     * @throws MProcRuleException
     */
    void updateMessageDestAddrTon(MProcMessage message, int newDestTon) throws
MProcRuleException;


    /**
     * Updating of destination address message NPI. In case of bad value (<0 or
>6)
     * MProcRuleException will be thrown
     *
     * @param message
     * @param newDestNpi
     * @throws MProcRuleException
     */
    void updateMessageDestAddrNpi(MProcMessage message, int newDestNpi) throws
MProcRuleException;

    /**
     * Updating of destination address message digits. Value can not be null and
must have length
     * 1-21 characters. In case of bad value MProcRuleException will be thrown
     *
     * @param message
     * @param newDigits
     * @throws MProcRuleException
     */
    void updateMessageDestAddr(MProcMessage message, String newDigits) throws
MProcRuleException;
```

```
    /**
     * Updating of source address message TON. In case of bad value (<0 or >6)
     * MProcRuleException will be thrown
     *
     * @param message
     * @param newDestTon
     * @throws MProcRuleException
     */
    void updateMessageSourceAddrTon(MProcMessage message, int newDestTon) throws
MProcRuleException;

    /**
     * Updating of source address message NPI. In case of bad value (<0 or >6)
     * MProcRuleException will be thrown
     *
     * @param message
     * @param newDestNpi
     * @throws MProcRuleException
     */
    void updateMessageSourceAddrNpi(MProcMessage message, int newDestNpi) throws
MProcRuleException;

    /**
     * Updating of source address message digits. Value can not be null and must
have length
     * 1-21 characters. In case of bad value MProcRuleException will be thrown
     *
     * @param message
     * @param newDigits
     * @throws MProcRuleException
     */
    void updateMessageSourceAddr(MProcMessage message, String newDigits) throws
MProcRuleException;

    /**
     * Updating of message text. Value must not be null and must have length 0-
4300. In case
     * of bad value MProcRuleException will be thrown
     *
     * @param message
     * @param newShortMessageText
     * @throws MProcRuleException
     */
    void updateShortMessageText(MProcMessage message, String
newShortMessageText) throws MProcRuleException;

    /**
     * Updating of UDH binary content. Value can be null or must have length >
0. In case
     * of bad value MProcRuleException will be thrown
```

```
     *
     * @param message
     * @param newShortMessageText
     * @throws MProcRuleException
     */
    void updateShortMessageBin(MProcMessage message, byte[] newShortMessageBin)
throws MProcRuleException;

    /**
     * Updating of ScheduleDeliveryTime - the time before which a message will
not be
     * delivered. This value can be null, this means that the message will be
tried to
     * delivery immediately. This value must be at least 3 hours before a
delivery
     * period end. If you pass the value that is later then 3 hours before a
delivery
     * period end, then 3 hours before a delivery period end will be set. If you
     * change both ValidityPeriod and ScheduleDeliveryTime values, then you have
to
     * setup ValidityPeriod value firstly.
     *
     * @param message
     * @param newScheduleDeliveryTime
     */
    void updateScheduleDeliveryTime(MProcMessage message, Date
newScheduleDeliveryTime);

    /**
     * Updating delivery period end time. This value can be null, this means
that
     * delivery period will be set to a default delivery period value of SMSC
GW.
     * If the value is more than max validity period that is configured for SMSC
GW,
     * then max validity period will be used instead of a provided value. If you
change
     * both ValidityPeriod and ScheduleDeliveryTime values, then you have to
setup
     * ValidityPeriod value firstly.
     *
     * @param message
     * @param newValidityPeriod
     */
    void updateValidityPeriod(MProcMessage message, Date newValidityPeriod);

    void updateDataCoding(MProcMessage message, int newDataCoding);

    void updateDataCodingGsm7(MProcMessage message);

    void updateDataCodingGsm8(MProcMessage message);
```

```
    void updateDataCodingUcs2(MProcMessage message);

    void updateNationalLanguageSingleShift(MProcMessage message,
        int newNationalLanguageSingleShift);

    void updateNationalLanguageLockingShift(MProcMessage message,
        int newNationalLanguageLockingShift);

    void updateEsmClass(MProcMessage message, int newEsmClass);

    void updateEsmClass_ModeDatagram(MProcMessage message);

    void updateEsmClass_ModeTransaction(MProcMessage message);

    void updateEsmClass_ModeStoreAndForward(MProcMessage message);

    void updateEsmClass_TypeNormalMessage(MProcMessage message);

    void updateEsmClass_TypeDeliveryReceipt(MProcMessage message);

    void updateEsmClass_UDHIndicatorPresent(MProcMessage message);

    void updateEsmClass_UDHIndicatorAbsent(MProcMessage message);

    void updatePriority(MProcMessage message, int newPriority);

    void updateRegisteredDelivery(MProcMessage message, int
newRegisteredDelivery);

    void updateRegisteredDelivery_DeliveryReceiptNo(MProcMessage message);

    void updateRegisteredDelivery_DeliveryReceiptOnSuccessOrFailure(MProcMessage
message);

    void updateRegisteredDelivery_DeliveryReceiptOnFailure(MProcMessage
message);

    void updateRegisteredDelivery_DeliveryReceiptOnSuccess(MProcMessage
message);

    // new message posting section
    /**
     * Creating a new message template for filling and sending by
postNewMessage() method
     */
    MProcNewMessage createNewEmptyMessage(OrigType originationType);

    MProcNewMessage createNewCopyMessage(MProcMessage message);

    MProcNewMessage createNewResponseMessage(MProcMessage message);
```

```
    /**
     * Posting a new message. To post a new message you need: create a message
template
     * by invoking of createNewMessage(), fill it and post it be invoking of
     * postNewMessage(). For this new message no mproc rule and diameter request
will
     * be applied.
     */
    void postNewMessage(MProcNewMessage message);
}
```

- `PostImsiProcessor` interface content:

```
package org.mobicents.smsc.mproc;

import org.apache.log4j.Logger;

public interface PostImsiProcessor {

    // access to environmental parameters
    /**
     * @return the logger that an application can use for logging info into
server.log
     */
    Logger getLogger();

    // actions
    void dropMessages();
}
```

- `PostDeliveryProcessor` interface content:

```
package org.mobicents.smsc.mproc;

import org.apache.log4j.Logger;

public interface PostDeliveryProcessor {

    // access to environmental parameters
    /**
     * @return the logger that an application can use for logging info into
server.log
     */
    Logger getLogger();

    boolean isDeliveryFailure();

    // actions
    /**
     * Creating a new message template for filling and sending by
postNewMessage() method
     */
    MProcNewMessage createNewEmptyMessage(OrigType originationType);

    MProcNewMessage createNewCopyMessage(MProcMessage message);

    MProcNewMessage createNewResponseMessage(MProcMessage message);

    /**
     * Posting a new message. To post a new message you need: create a message
template
     * by invoking of createNewMessage(), fill it and post it be invoking of
     * postNewMessage(). For this new message no mproc rule and diameter request
will
     * be applied.
     */
    void postNewMessage(MProcNewMessage message);

}
```

11. Once all the source code is fully implemented, user should deploy it.
    - User must compile java code and create a jar library. Copy the created jar into the folder *restcomm-smscgateway-<version>/jboss-5.1.0.GA/server/<profile>/deploy/restcomm-smsc-server/lib*.
    - You need to update *restcomm-smscgateway-<version>/jboss-5.1.0.GA/server/<profile>/deploy/restcomm-smsc-server/META-INF/jboss-beans.xml* config file.

      Define factory implementing `MProcRuleFactory`in above example it will be `MProcRuleFactoryTestImpl`:

```
    <bean name="MProcRuleFactoryTest"
class="org.mobicents.smsc.mproc.testimpl.MProcRuleFactoryTestImpl">
    </bean>
```

bean name can be some unique name.

In the "bean name="SmscManagement"" section - property <property name="MProcRuleFactories"> - after the line <inject bean="MProcRuleFactoryDefault"/> (that means default mproc rules implementing) - add the line that describes your new created bean:

```
        <property name="MProcRuleFactories">
            <list elementClass="org.mobicents.smsc.mproc.MProcRuleFactory">
                <inject bean="MProcRuleFactoryDefault"/>
                <inject bean="MProcRuleFactoryTest"/>
            </list>
        </property>
```

12. Start SMSC GW
13. Create an mproc rule by using CLI interface. Let's create a rule with id=1 that will be applied for messages which destination adresses are started with "22" and this rule will add prefix "33". To achive it run CLI console and run the following command

    smsc mproc add testrule 1 22 33

    where "testrule" is an implemented class name, "22 33" is aparameter string.

14. Send a message to the destination address that starts with "22" (for example "221111") and find that the message will be delivered to the address with prefix "33" ("33221111" in our example).| |

SMS Home Routing

According to traditional GSM specifications, all outbound messages pass through an SMSC in the sending network. This allows the SMSC to convert the Mobile Originated (MO) messages into Mobile Terminated (MT) messages and deliver them directly to the destination devices, regardless of what network they are on. As a result, inbound messages generated on other networks will be sent directly to the destination devices under the control of the SMSC in the sending network, not the home network. In this traditional setup, operators can add value to the MO messages but not to the MT messages that the customer may receive from other networks, since they do not pass through an SMSC in the home network.

SMS Home Routing is a modification to the original GSM specifications, adopted by the 3GPP in 2007, that changed the way inbound messages are treated by the mobile networks. Home Routing uses the recipient network's Home Location Register (HLR) to change the flow of inbound messages, directing them to an MT Services Platform, rather than straight to the destination

devices. The MT Services platform can add value to the MT messages and apply advanced services such as anti-spam, protection, divert, archiving, etc. to the messages prior to delivery.

Home Routing enables operators to offer a full range of value-added services to both inbound and outbound SMS thereby enhancing customer experience while generating additional revenue for the operators.

Restcomm SMSC supports SMS Home Routing that enables you to handle Mobile Terminated messages in the network that owns the customer so you can offer a full range of advanced and value added services on both inbound and outbound SMS. You need to configure SMSC GW before you can use SMS Home Routing. For more details refer to [_smsc_gateway_server_hr].

Interworking with Diameter OCS server

General information

SMSC GW can interoperate with an OCS server via Diameter protocol connection.

For this case SMSC GW can be configured so any incoming SMPP, MO SS7, Home Routing SS7 and SIP originated messages before next processing will be sent to OCS server. Then OCS server can check if SMSC GW must accept the message or not. If yes, SMSC GW will process the message, if not, the message will be dropped. OCS server can also make any charging operations as needed.

For SMPP and MO SS7 originated messages SMSC GW will return error (reject) in the response back to the orgiginated message submit message. So MO and SMPP part will be informed if the message is rejected. (This functionality is not implemented now for Home Routing SS7 and SIP originated messages).

For interconnecting with OCS Server SMSC GW uses diameter CCR (Credit-Control Request) with following filled AVPs (all of them are located in AVP Service-Information (873)):

Table 4. The AVP list

AVP name	AVP code	AVP Type	Parent AVP	Description
SMS-Information	2000	Grouped	Service-Information	SMS service specific information elements
Originator-SCCP-Address	2008	Address	SMS-Information	The CallingParty SCCP Address of MO message

AVP name	AVP code	AVP Type	Parent AVP	Description
SMSC-Address	2017	Address	SMS-Information	SMSC address as it is present in SM_RP_DA field in MO request. For not MO messages the value will be taken as a configured SMSC GT for networkId area
Data-Coding-Scheme	2001	Integer 32	SMS-Information	Data Coding Scheme of a message (for example 0 for GSM7 encoding or 8 for UCS2 encoding)
SM-Message-Type	2007			Message type, usually SUBMISSION (value = 0)
Originator-Interface	2009	Grouped	SMS-Information	Information related to the Interface on which the message originated.
Interface-Id	2003	UTF8-String	Originator-Interface	NetworkID value of the Interface on which the message originated.
Interface-Text	2005	UTF8-String	Originator-Interface	Name of ESME / SIP connector on which the message originated.
Recipient-Info	2026	Grouped	SMS-Information	Information associated with a recipient.
Recipient-Address	1201	Grouped	Recipient-Info	The address of message receiver (destination address)

AVP name	AVP code	AVP Type	Parent AVP	Description
Address-Type	899	Enum	Recipient-Address	Type of Recipient-Address: 1-MSISDN (for TON=1 - international) or 6-Other (for TON!=1)
Address-Data	897	UTF8-String	Recipient-Address	Digits of Recipient-Address
Originator-Received-Address	2027	Grouped	SMS-Information	The address of message sender (source address)
Address-Type	899	Enum	Originator-Received-Address	Type of Originator-Received-Address: 1-MSISDN (for TON=1 - international) or 6-Other (for TON!=1)
Address-Data	897	UTF8-String	Originator-Received-Address	Digits of Originator-Received-Address

When OCS server responds with 2001 code (access granted) SMSC GW will accept the message, all other respond codes will reject the message.

Configuring

You can use Telestax OCS server or any third party OCS server as you wish. SMSC GW party configuring is the same for both. Here there is a describtion for the case of Telestax OCS server and as an example - the simpliest case when we have OCS at the same host as SMSC GW.

Procedure: Steps for configuring of SMSC GW and OCS server

1. SMSC GW must be already configured for accepting and sending messages (general, SS7, SMPP, etc parts)
2. You can download for testing the Telestax OCS server from https://telestax.zendesk.com/forums/22947518-Product-Downloads. Download and unpack it to a separate folder.
3. SMSC GW uses the updated AVP dictionary library. Before using of OCS server copy from SMSC GW the file <smsc gw root folder>/resource/ocs/dictionary.xml to Telestax OCS server to the folder <jboss>/server/default/deploy/RestComm-diameter-mux-6.2.0.GA.sar/config.
4. for simple test configuring of SMSC GW and OCS server you can copy example config files jdiameter-config.xml

- from <smsc gw root folder>/resource/ocs/smsc-part to SMSC GW <jboss>/server/default/deploy/RestComm-diameter-mux-6.2.0.GA.sar/config
- from <smsc gw root folder>/resource/ocs/ocs-part to OCS server <jboss>/server/default/deploy/RestComm-diameter-mux-6.2.0.GA.sar/config

5. Those configs suppose that you use Telestax OCS server and it is on the same host. If not you need to make extra configuring. In the testing config files SMSC GW plays a role of a diameter client and OCS server - a diameter server. In this case for OCS server part we need to update jdiameter-config.xml file:
   - <LocalPeer> section with URI, IPAddresses and Realm.
   - In <Network> we need to specify <Peer> (SMSC GW diameter IP) and <Realm> (name and the same Realm that is configured in SMSC GW part)

     For SMSC GW part we need to update:

   - <LocalPeer> section with URI, IPAddresses and Realm.
   - In <Network> we need to specify <Peer> (OCS server diameter IP) and <Realm> (name and the same Realm that is configured in OCS server part) You can read more for configuring in the manuals for Telestax Diameter protocol and Telestax OCS server.
6. Start OCS server. If it is located in the same host as SMSC GW you need to run it "-b 127.0.0.2" parameter so it uses 127.0.0.2 IP address (127.0.0.1 is used by SMSC GW) Use fo starting the command like:

   ```
   run -b 127.0.0.2
   ```

7. For accessing OCS managing console (after OCS server is started) use a browser with URL: http://127.0.0.2:8080/charging-server-rest-management/#/users

   Avoid of using of Microsoft Internet Explorer, OCS server does not work correctly with it.

   Add one or more test subscribers by the console. You need to specify an address of message sender (source address). OCS Server idendifies a subscriber by its sender address.

8. Start SMSC GW
9. You need to configure Destination Realm, Destination Host, Destination Port and User Name (of OCS server) - see the chapter "6.1.5. Diameter Settings" of SMSC GW Admin guide. You can take values from jdiameter-config.xml. You can assign for User Name any value (for example "telestax").
10. You need to specify which messages will be sent to OCS server (see chapter [_smsc_diameter_settings]). For example if you want to charge mobile originated messages - specify "Mobile Originated SMS Charged" to value "diameter".
11. SMSC GW is configured

CDR Logging Settings

Restcomm SMSC is configured to generate CDR in a plain text file located at *restcomm-smscgateway-<version>/jboss-5.1.0.GA/server/<profile>/log/cdr.log* and also detailed CDR in Cassandra table `MESSAGES_yyyy_mm_dd`.

Restcomm SMSC can be configured to generate CDR for different message processing modes (`datagramm, transactional, storeAndForward`) and also for both receipt and regular messages or generate CDR only for regular messages. This is configurable by setting `generateCdr`, `generateArchiveTable` and `generateReceiptCdr` using the CLI or the GUI. For more details refer to Sections [_generatecdr], [_generatearchivetable] and [_generatereceiptcdr].

The CDR generated in a text file is of a specific format. The details of the format and the possible values for the fields recorded in the CDR log file are explained in [_monitoring_smsc_cdr_log].