Mobile SMS Services Configuration

Since 4.2.2 release, the hybris Commerce Suite offers the integrated Short Message Service(SMS) functionality. This feature allows you to set up an additional channel for interaction with your customers. It is built on top of the high performance and resilient Action Framework. The SMS services provide a trouble-free performance and easy integration with custom business logic.

In this document you find information on usage of mobile SMS services, configuration of one- and two-way actions.

Aggregator
The company that provides the infrastructure to deliver messages to end users.

Shortcode

The numeric destination that the end user uses to send messages to the hybris Commerce Suite. Examples are: 12345, 66899, and so on. The shortcodes are country specific.

CDMA/GSM networks

Mobile networks. See Wikipedia on Cellular Networks and Wikipedia on GSM Standard for more details.

WAP push

Technology providing access to downloadable content hosted on a web server, Java applications, images, polyphonic ringtones, videos, and so on. The operation is based on an XML structure containing the address or URL of the content, which is compiled and sent as a binary SMS.

Keyword

The text that the user has to send to the shortcode to trigger a response. Example:

HYBRIS VOUCHER.

SMPP

Short Message Peer to Peer Protocol. See Wikipedia on SMPP

SMS Configuration Overview

The block diagram below outlines the main elements you need to set-up an SMS service within the hybris Commerce Suite.

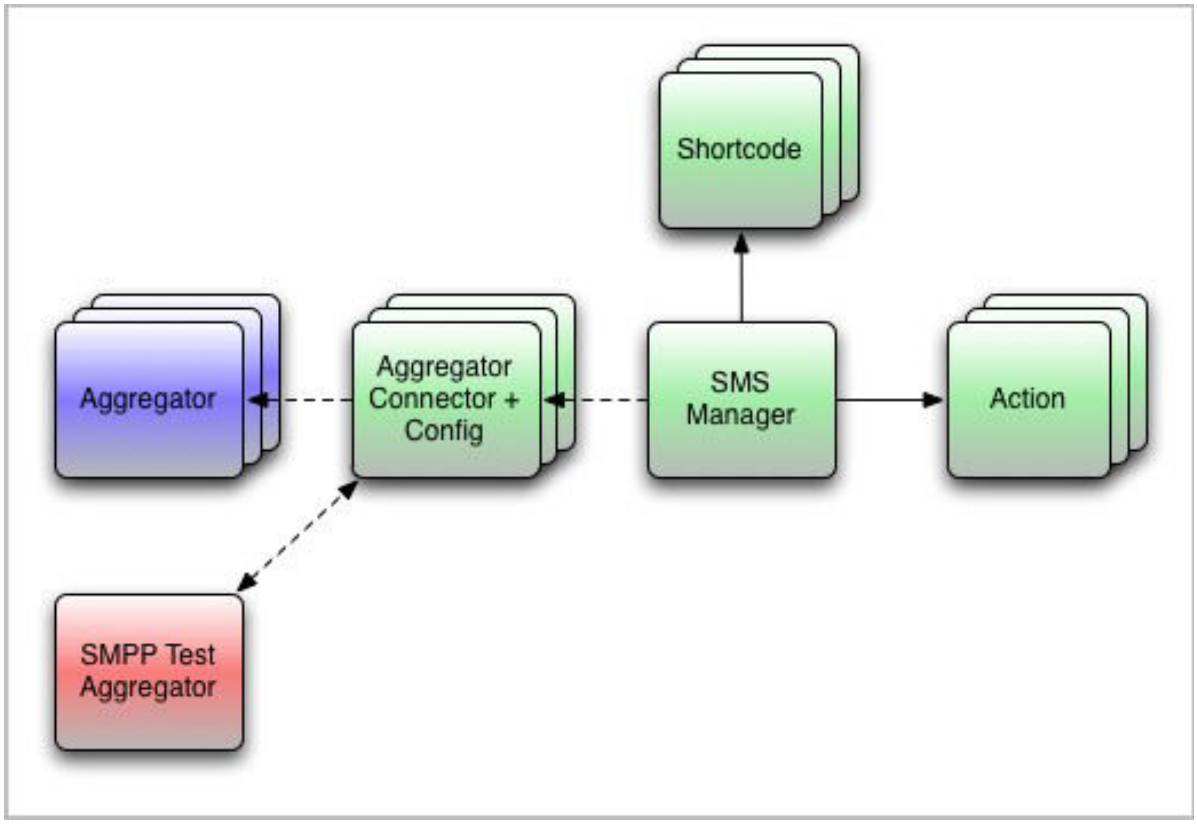


Figure: A diagram of mobile SMS configuration.

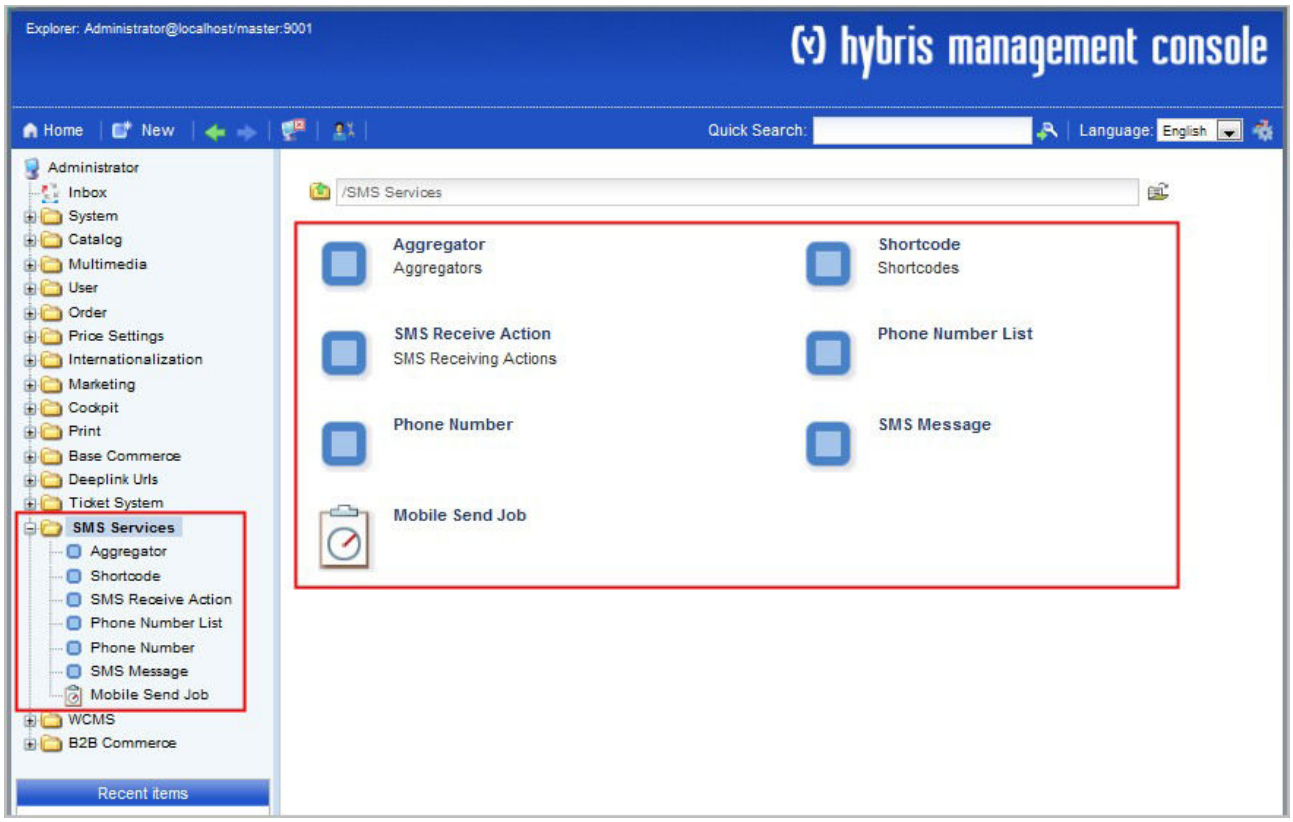
The blocks in green are the elements that are managed within hybris Commerce Suite(hMC), blue elements are external to the hybris system and the elements which are red are used to test, set up and troubleshoot, but those are not needed in the production environment.

- The **aggregator** is an external company that takes care of the SMS delivery from and to the carriers.
- The **aggregator connector** within a hybris system takes care of implementing the protocol needed for message exchange with an aggregator. This protocol is typically SMPP.
- The **SMS manager** is an abstract concept that outlines that hybris Commerce Suite has logic that based on a message and configuration (shortcode), triggers an action. This concept means also that a system manages the load, persistence, error handling, and so on.

- **Shortcode** is the element where all the SMS-related configuration is configured, that is: keywords, shortcodes, and so on.
- **Actions** are the logic element in the hybris Commerce Suite where a specific action is performed such as **creating a voucher**, **sending a link**, and so on.
- **SMPP test aggregators** are external programs that simulate being an aggregator to a hybris system. These elements are used to configure a system without really needing a contract with an aggregator. Example of such program is **SMPPSim**.

SMS Management Within the hMC

ou find all SMS-specific management options in the **SMS Services**node in the hMC:



SMPP Connector Configuration

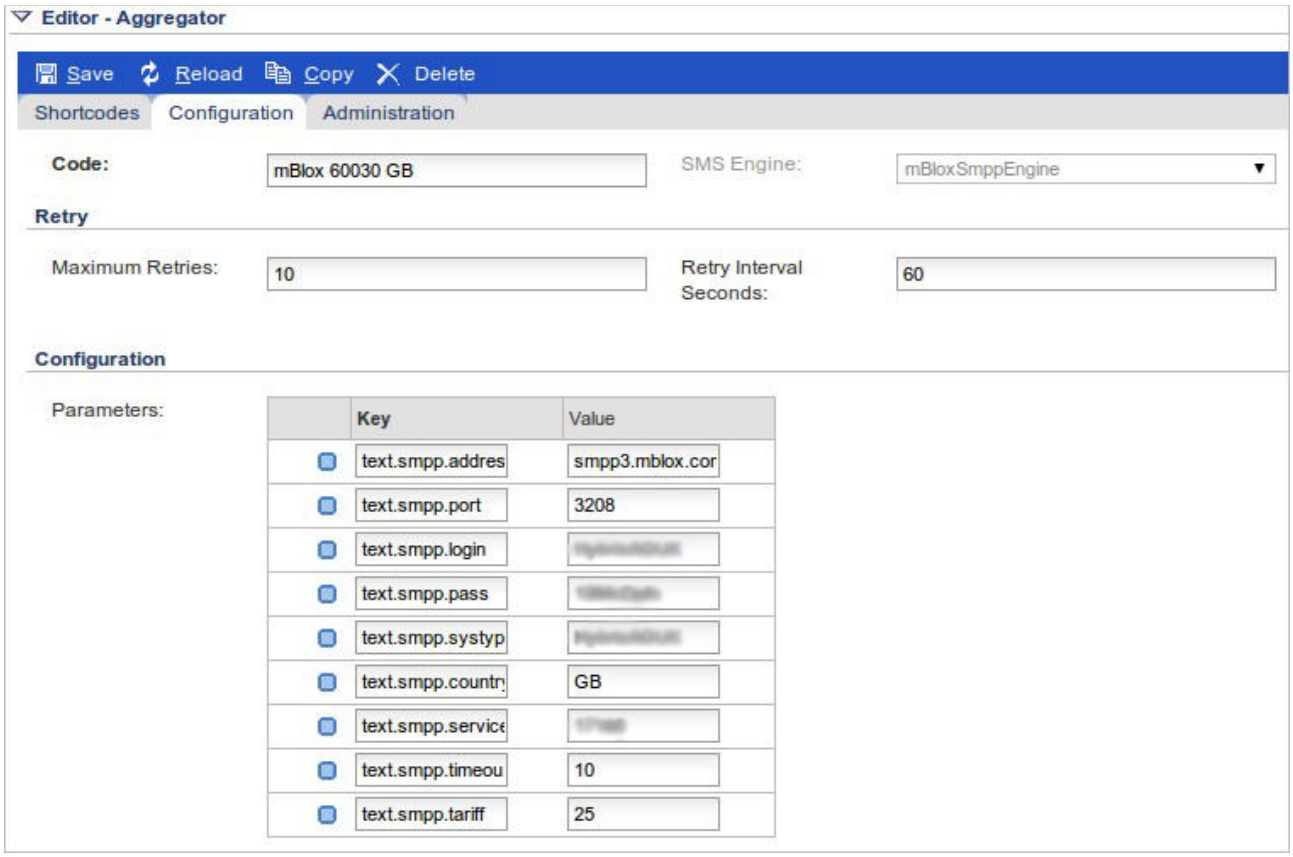
The connector is a piece of software responsible for maintaining the connection to and from the aggregator, that means sending and receiving the user messages. Although typically every aggregator in the market has custom HTTP/XML interface, there is a standard protocol called SMPP, supported by most of them. The hybris

Commerce Suite provides a standard SMPP connector that you can customize for your needs.

You can find the configuration options of a connector in **Configuration** tab of the **Aggregator** node in the hMC. To add a property, right-click the area of **Parameters** and select **Create Aggregator Parameter**. After providing all parameters, click the **Save** button. The system persists changes.

The supplied SMPP connector needs following properties to be configured in order to connect to the aggregator:

text.smpp.address	The IP of the SMPP aggregator systems.
text.smpp.port	The port of the SMPP aggregator systems.
text.smpp.login	Credentials for SMPP connection
text.smpp.pass	Credentials for SMPP connection.
text.smpp.systype	Used by aggregators to identify the connection for billing or administrative purposes.
text.smpp.servicetype	Used by aggregators to identify the connection for billing or administrative purposes.
text.smpp.timeout	Elapsed time between SMPP enquire_link messages to keep the connection alive.
text.smpp.country	The ISO code of the country for which the aggregator provides connection.
text.smpp.tariff	Used by aggregators to identify the connection for billing or administrative purposes



Configuring SMPP Gateway for European Languages

By default, SMPP gateway uses GSM 3.38 encoding which may cause some encoding issues in countries such as Spain, Germany or Poland, which use non-default diacritic symbols. In theory the default character set supports characters such as or , but tests proved to render improper results. It is recommended to change the SMS encoding if messages are going to be delivered in countries other than the United Kingdom.

The SMPP driver accepts the following encoding:

- 0: GSM 3.38 (default)
- 1: ASCII
- 3: ISO-8859-1
- 4: Binary encoding GSM 3.38 (data coding set to 4)
- 8: ISO-10646-UCS-2, which is an archaic predecessor of UTF-16.

It is recommended to use the value **8** for European encoding. In order to set the aggregators encoding set a new property with text text.smpp.encoding and value 8. If it is present, update the value for the encoding key.

SMPP Parameters Reference

This section describes parameters to be used for SMPP driver.

Aggregator Level Parameters

text.smpp.address	*defaultsmmhost*	SMPP host IP/address.
text.smpp.port	*2775*	SMPP port
text.smpp.login	*login*	SMPP SystemID
text.smpp.pass	*password*	SMPP password
text.smpp.systype	*system*	SMPP System Type. Consult your SMPP documentation.
text.smpp.servicetype	*12345*	SMPP Service Type. Consult your SMPP documentation
text.smpp.tariff	**none**	SMPP Tariff
text.smpp.encoding	*0*	SMPP encoding. Default: GSM 3.38.
text.smpp.country	**none**	ISO code of the country the service is set up for.
text.smpp.timeout	*30*	Network connection timeout in seconds.
text.smpp.retrydelay	*30*	Send retry delay when an error occurs, in seconds.

Node Parameters Customization

You can customize node parameters in the**mobileservices**extension**project.properties**file.

text.smpp.enabled	*false*	Set to true to enable SMPP networking.
text.smpp.mode	*-1*	SMPP mode, by default set to **transceiver** mode.

Error Codes Reference

This section provides lists of error codes generated by system.

Mobile Message Error Codes

INVALID_PHONE_NUMBER	Phone number does not validated for the country in use.
MAX_SIZE_EXCEEDED	Message size exceeds configured message size (in characters).
UNSUBSCRIBED	Reserved for future use.
NOROUTE	There is no route for either receiving or sending.
ACTIONMISSING	Unused
LINKNOTSUPPORTED	Link used in CDMA aggregator.
FILTERED	Message reception or delivery unauthorized by a list (blacklist or testlist).
WRONGCONFIG	Output engine ID not present in the system.
UNKNOWN	Other errors

Mobile Aggregator Error Codes

IO_ERROR	Reserved for future use.
REJECTED	Rejected by aggregator. It can be a temporary error in which case the error is cleaned up if the sending succeeds.
NO_CREDIT	Reserved for future use.
LIMIT_EXCEEDED	Reserved for future use.
UNAVAILABLE	No delivery route available. It can be a temporary error in which case the system retries and finally the error is cleaned up.
UNKNOWN	Other errors

Mobile SMS Services

The hybris**mobileservices**extension contains components responsible for sending and receiving SMS messages to the hybris system, sending SMS to customers, and responding to the SMS messages received from customers.

It consists of:

- A server side module for sending, receiving, and processing SMS messages.
- A UI integrated inside the hybris Management Console(hMC), which can be used to configure the SMS use cases of your organization.

Definitions

SMS	A mobile short text message.

WAP	Wireless Application Protocol. A technology framework for mobile pages.
WAP Markup	Markup language for mobile devices. More common are WML or XHTML MP.
WML	WAP 1.x markup language
XHTML MP	WAP 2.x markup language
Two-way message	A message sent by hybris system as a response to a received SMS.
One-way message	A message sent by hybris system. The triggering of the message is not initiated by the reception of a SMS. It is triggered by system according to settings.
WAP Push	A technology to send a link to a phone supporting WAP.
Strategy Pattern	The ability to select a specialized strategy object for a particular task in run time .
Action Framework	A hybris technology for executing spring beans with run-time parameters and database driven configuration. Spring beans must implement the **ActionPerformable** interface and model configuration must extend **AbstractAction**.
Process Engine	A hybris technology that allows the asynchronous execution of distributed code in the hybris cluster. It supports event triggering, scheduling, error recovery and tolerance. The SMS services are built on top of this technology.

Overview of the Server Side Module

The server-side SMS components use two hybris technologies called the hybris Process Engine and Action Framework.

The Process Engine allows the SMS code to be spread on the hybris cluster, takes care of failures and retrial in case of error, and supports scheduled execution of program retries or deliveries.

The Action Framework is a standard hybris way to plug user generated components into the system.

The SMS Services have built-in load management. For more information on load management within a hybris system, refer to Mobile SMS Services Configuration document, **Description of Load Management** section.

A strong knowledge of the Action Framework is mandatory to personalize the shipped services or build new ones. The developers of SMS services do not need to be aware of how the Process Engine works, but it is highly advisable that they do. You find several examples of SMS actions provided in this document.

One-Way and Two-Way Messaging

The SMS services functionality supports both one-way and two-way messaging.

One-way messaging is the one where the process is triggered by the hybris system. You can use it to send all kinds of interesting information such as promotions, content, and so on, to the customer without the customer triggering it. One-way messaging uses short codes and aggregators defined.

Two-way messaging is the one produced as a response to a received SMS. Typically the system receives a message, processes it and a response is created. Then it is sent to the customer. Two-way messaging uses aggregators, short codes, actions and assignments (binding of an action to a short code using a keyword).

Standard Messaging and Link Messaging

The hybris Commerce Suite supports both normal plain text messages or WAP Push indicators. WAP Push messages are a standard way of sending links to WAP pages that work only in GSM-based networks. Although all devices which support WAP 1.x or 2.0 should accept these messages, in practice some devices and manufacturers don't support it. This is the case of the popular Apple iPhone mobile phone.

To respond to this eventuality, you can disable WAP Push by configuration to stop sending WAP Push messages from the system. When disabled, all WAP Push messages are converted to plain text messages, which the text being a combination of

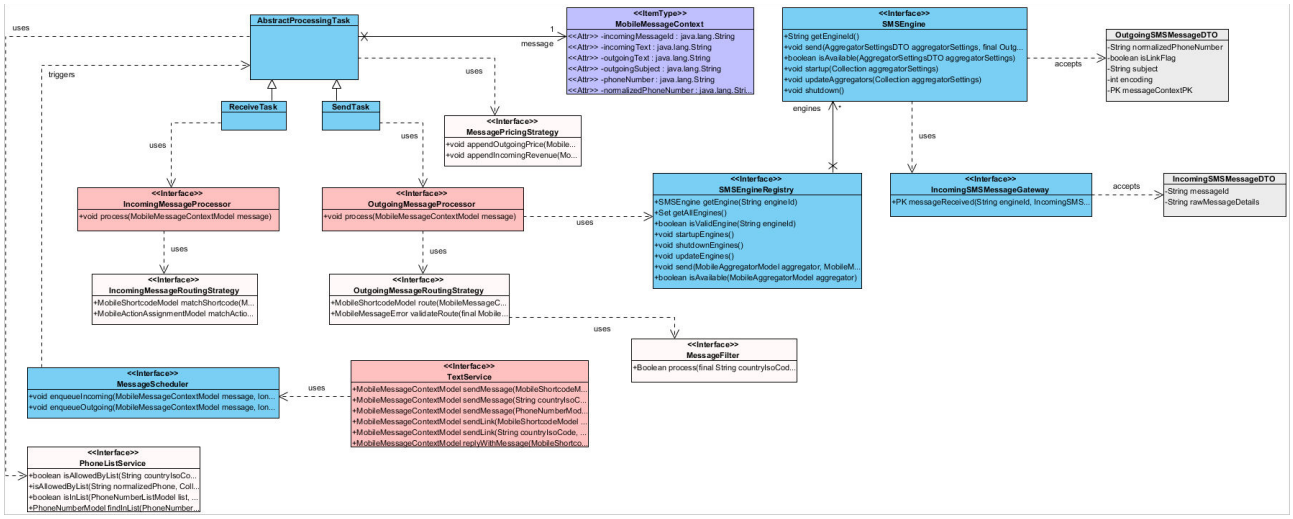
the WAP Push subject and payload URL. This behavior is controlled by the parameter **mobile.wappush.enabled**, which is disabled by default .

Validation

Every phone number that enters the system, either using the external aggregator or the hMC, is validated for correctness against a database of valid phone number formats per country. Phone numbers which are not valid, for example, lacking a digit, and so on are rejected by the system on input.

SMS Life Cycle

There are two connecting points between SMS services and the external world - incoming gateways and outgoing gateways. The life cycle of a message varies depending on if it is one- or two-way messaging.



Incoming Gateways

Incoming gateways are input sources that inject SMS into the system and typically use the**IncomingSMSMessageGateway**interface or make use of the**DefaultIncomingSMSMessageGateway**bean.

Outgoing Gateways

Outgoing gateways send messages to the aggregators or external systems. They must implement the interface**SMSEngine**.

One-Way Messaging Life Cycle

For one-way messaging the message is either injected into the system via**textService.sendMessage**or**textService.sendLink**in each case there is only one asynchronous execution step, and it is the message delivery step, identical to the two-way scenario.

The other possibility is that the message is sent as a result of executing any of the one-way actions defined, which you can potentially execute asynchronously and generate a call to**textService.sendMessage**or**textService.sendLink**. In this case, there are two steps: a processing step and a delivery step.

The delivery step is identical to two-way message delivery. Message processing is also similar but there's one important difference. When processing actions in two-way scenario, a response is generated by calling**textService.replyWithX**, while one-way generates a standalone message via**textService.sendX**.

There is nothing in the system that prevents a gateway of operating as receiving and sending gateway. This is the case of the SMPP gateway.

Both type of gateways use the concept of**engineId**. Each gateway, incoming or outgoing, is identified by a string that is used both for tracing where the message was injected into the system and to decide which particular implementation the aggregator uses.**EngineId**are declared as an enumerated type in themobileservices-items.xmlfile.

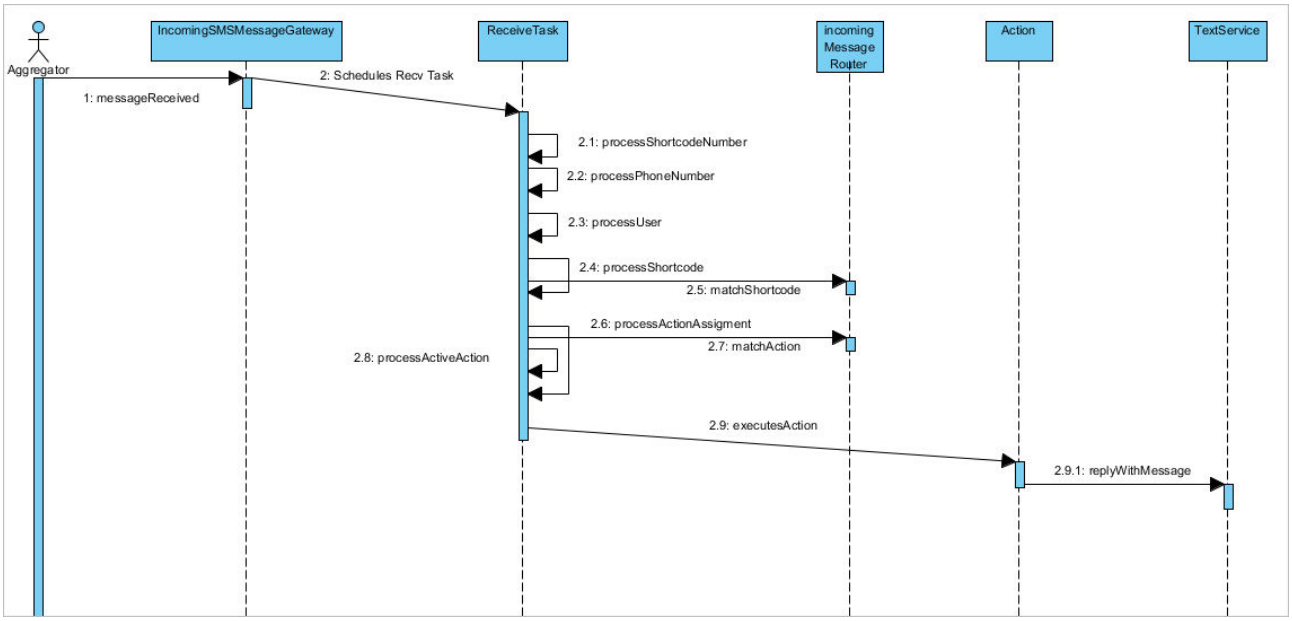
mobileservices-items.xml

```
...
<enumtype code="EnginesType" autocreate="true" generate="true"
dynamic="true">
  <value code="smppEngine" />
  <value code="mBloxSmppEngine" />
  <value code="bulkSMSEngine" />
  <!-- other engines to be included here -->
< ... >
</enumtype>
...
```

Gateways use input/output DTOs to encapsulate and hide the message model to the gateway implementations.

Two-Way Messaging Life Cycle

Two-way messaging is an asynchronous activity that takes place in three steps. Each step is executed asynchronously and can take place anywhere in the hybris cluster. The steps involved are: message reception, processing, and delivery.



Message Reception

Message reception means that the message injected by the incoming gateway is received by the system. The system looks up the dictionary of available keywords, settings, and actions for the short code the message is using and decides on the optimal routing strategy. That means, it decides which action is going to be executed in the processing step and which engine is going to be used in the delivery step.

Message reception is initiated by the**DefaultMessageScheduler**. This class queues messages for receiving or sending and delegates them to the**Receive**task, which is the one that is executed asynchronously.

The**Receive**task uses the**DefaultIncomingMessageProcessor**to route the message basing on its keyword and properties. The routing takes place in

the**IncomingMessageRoutingStrategy**, which follows the strategy design pattern and you can swap it for a customized routing strategy of your own.

The incoming message processor finally delegates the action that generates the response, and therefore triggers the processing step, to the Action Framework to execute. The action that is matched by the previous process gets all the message information, original phone number, message, keyword, and so on. With that information it is the responsibility of the action how to proceed.

Message Processing

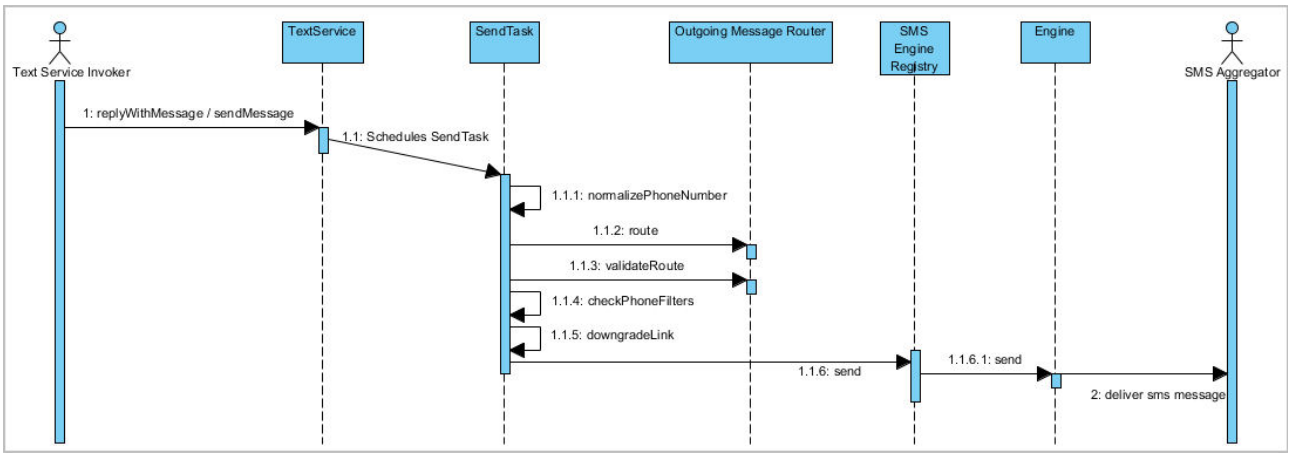
Message processing means that an action is executed. Once the right action to execute is identified as explained in the previous section, the next step consists of executing the code that generates the response to the customer, that is, executing the hybris action. Actions are JavaBeans that can use the**TextService**API to signal a response to the incoming data, but that do not modify or alter the message object in any way.

When invoking the**TextService**to generate a response, the system queues the response to be executed by the**SendTask**task. It triggers the third asynchronous step.

Message Delivery

Message delivery, as initiated by the message scheduler bean, is responsible for performing additional final validation on the message. The phone number is validated. Then an output route is chosen by the outgoing message routing strategy. The blocking lists are verified to ensure the destination number is not blocked on the short code. Next, the message is converted, if necessary, from WAP Push to plain text and finally passed to the**SMSEngine**registry which identifies the engine responsible for sending based on the aggregator**engineId**.

The**SMSEngine**is the one responsible for connecting to the external system, sending the messages using the appropriate protocol and taking care of aggregator negotiation and error detection.



mobileservices Extension APIs

The SMS services are made of a number of classes and APIs. The ones that are relevant for the technical user are explained in this section.

TextService

The**TextService**is the service that takes care of sending replies to customers. As such is the service that the hybris actions will use to reply.

getExclusiveShortCode	Looks up an exclusive short code using its unique code.
getSharedShortCode	Looks up a shared short code by a specified code and keywords.
getAvailableShortcodesForSending	Locates all short codes for the specified country which support sending outgoing messages. Caution Note that the final routing decision is being done by the **OutgoingMessageRoutingStrategy** which is currently set up in the mobile spring configuration. This method simply lists all potential ones.
sendMessage	Sends a text SMS message to the specified number. The specified **MobileShortcodeModel** is used to

	obtain the origin number of the message. Messages are not sent immediately but are queued. Therefore the returned model shows intermediate states. Use **MessageHelper#blockUntilProcessed** if you must wait until sending is done.
sendLink	Sends a WAP Push message to the specified number. The specified **MobileShortcodeModel** is used to find out the origin number. Messages are not sent immediately but are queued. Therefore the returned model shows intermediate states.
replyWithMessage	While processing an incoming message, this methods allows to send a reply text message using the incoming message information. The specified **MobileShortcodeModel** is used to find out the origin number. Reply messages are not sent immediately but are queued. Therefore the returned model shows intermediate states. Use **MessageHelper#blockUntilProcessed** if you must wait until sending is done.
replyWithLink	While processing an incoming message this methods allows to send a reply link message using the incoming message information. The specified **MobileShortcodeModel** is used to send the message on behalf. Reply messages are not sent immediately but are queued. Therefore the returned model shows intermediate states. Use **MessageHelper#blockUntilProcessed** if you must wait until sending is done.

done	While processing an incoming message this method allows to mark the message as done. Such a message is considered as processed without an error.The mobile framework is allowed to either delete or archive it from now. Since message processing is done by custom **AbstractActionModel** actions it is recommended that implementors signal the end of processing by calling either **#done** or **#discard** in case no reply is being triggered.
discard	While processing an incoming message, this method allows to mark the message as discarded. It further allows optionally to specify an error type, a message and even a causing exception to be specified. Such a message is considered as processed abnormally. Therefore the mobile framework may keep such messages for reporting reasons. Since message processing is done by custom **AbstractActionModel** actions it is recommended that implementors signal the end of processing by calling either **#done** or **#discard** in case no reply is being triggered.

Task1:- In this Task we are going to Create One Way Message Service When Customer Register in our site he will Receive the SMS

Step 1 :- Make sure Mobile attribute mandatory field in registration page.

registration.tag
<formElement:formInputBox idKey= register.mobileNumber labelKey= register.mobileNumber path= mobileNumber inputCSS= text mandatory= true />

< Your Extention>facades-beans.xml
<bean class= de.hybris.platform.commercefacades.user.data.CustomerData > <property name= mobileNumber type= String /> </bean>

Now we Need to Store This MobileNumber in Database By Using ReversePopulator (We Know How to write ReversePopulator so we are not Mention here again)

Step 2 :- Take sms engine configration from client and according that api create SMS Engine.

Local.properties (Config)
#TestSendSMSEngine configration ## text.http.url =http://ultrapro.aonesms.com/API/SendMsg.aspx text.http.uname =20150147 text.http.pass =123456 text.http.send =VVKart

Note:-
Before Doing this Make sure that **MobileService** Extention is Active or not If it is not add in **localextetntion.xml** and also add in **extenioninfo.xml** of extention where Your are written bellow listed Calssess and actions.Do **ant all** and **ant Updatesystem**

Step 3:-Creating our own sms engine class in facade inside that change the method in which URL is creating.

TestSendSMSEngine.java

```
import de.hybris.platform.mobileservices.text.engine.AggregatorSettingsDTO;
import de.hybris.platform.mobileservices.text.engine.OutgoingSMSMessageDTO;
import de.hybris.platform.mobileservices.text.engine.SMSEngineException;
import de.hybris.platform.mobileservices.text.engine.impl.AbstractSMSEngine;
import de.hybris.platform.mobileservices.text.util.SendSMSHelper;
import de.hybris.platform.util.Config;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.lang.StringUtils;
import org.apache.log4j.Logger;


/**
 * @author rehaman
 *
 */
public class TestSendSMSEngine extends AbstractSMSEngine
{

	private static final Logger LOG =
Logger.getLogger(TestSendSMSEngine.class.getName());
	public static final String SERVER_PARAMETER = "text.http.url";
	public static final String UNAME_PARAMETER = "text.http.uname";
	public static final String PASS_PARAMETER = "text.http.pass";
	public static final String SENDER_PARAMETER = "text.http.send";

	/*
```

```
	* (non-Javadoc)
	*
	* @see
de.hybris.platform.mobileservices.text.engine.SMSEngine#send(de.hybris.platform.
mobileservices.text.engine.
	* AggregatorSettingsDTO,
de.hybris.platform.mobileservices.text.engine.OutgoingSMSMessageDTO)
	*/private String readRemoteData(final String target) throws IOException
	{
		final StringBuffer result = new StringBuffer(4192);
		InputStreamReader inStream = null;
		BufferedReader buff = null;


		try
		{

			LOG.info("****************************************Now
is trying to estiblesh the connection **********************");
			final URL url = new URL(target);
			final URLConnection urlConn = url.openConnection();
			inStream = new InputStreamReader(urlConn.getInputStream());
			buff = new BufferedReader(inStream);

			for (String line = buff.readLine(); line != null; line =
buff.readLine())
			{
				result.append(line);
			}

		}
		catch (final MalformedURLException e)
		{
			LOG.error("Wrong URL " + target, e);
			return null;
		}
		finally
		{
			if (buff != null)
			{
				try
				{
					buff.close();
```

```
                        }
                        catch (final Exception localException3)
                        {
                                //
                        }

                }
                else if (inStream != null)
                {
                        try
                        {
                                inStream.close();
                        }
                        catch (final Exception localException4)
                        {
                                //
                        }
                }
        }

        return result.toString();
    }


    private String getHexEncoding(final byte[] bytes)
    {
        final StringBuffer result = new StringBuffer();
        for (int i = 0; i < bytes.length; ++i)
        {
            result.append(Integer.toString((bytes[i] & 0xFF) + 256,
16).substring(1));
        }
        return result.toString();
    }

    protected String buildUrl(final String serverUrl, final Map<String, String>
parameters) throws UnsupportedEncodingException
    {
        LOG.info("************************URL Creating
**************************************");
        final StringBuilder result = new StringBuilder(serverUrl);
        if (!(StringUtils.contains(serverUrl, '?')))
        {
```

```
                result.append('?');
            }
            for (final Map.Entry e : parameters.entrySet())
            {
                if (StringUtils.isEmpty((String) e.getValue()))
                {
                    continue;
                }
                if ((!(StringUtils.endsWith(result.toString(), "?"))) && (!
(StringUtils.endsWith(result.toString(), "&"))))
                {
                    result.append('&');
                }
                result.append((String)
e.getKey()).append('=').append(URLEncoder.encode((String) e.getValue(), "ISO-
8859-1"));
            }
            LOG.info("***********final url is " + result.toString());
            return result.toString();
        }

        protected boolean parseGatewayResponse(final String response)
        {
            LOG.info(" ***********parsing response getway**************");

            return ((response.startsWith("0|")) || (response.startsWith("1|")));
        }

        /*
         * protected boolean sendLink(String normalizedPhoneNumber, String
subject, String url, AccountParams accountParams)
         * { String text = SendSMSHelper.removeSpecialCharacters(subject); try
{ byte[] payload =
         * SendSMSHelper.generateWapPush(url, subject); String encodedMessage =
getHexEncoding(payload); HashMap params = new
         * HashMap(); params.put( username , accountParams.login);
params.put( password , accountParams.password);
         * params.put( message , encodedMessage); params.put( dca , 8bit );
params.put( msisdn , normalizedPhoneNumber);
         * return
parseGatewayResponse(readRemoteData(buildUrl( accountParams.server,
params))); } catch (Exception ex) {
         * LOG.error( Error sendind SMS link to [ + url + ] with subject[ + text + ] to
```

```
+ normalizedPhoneNumber, ex); }
         * return false; }
         */
        protected boolean sendSms(final String normalizedPhoneNumber, final
String inputText, final AccountParams accountParams)
        {
                final String text =
SendSMSHelper.removeSpecialCharacters(inputText);
                try
                {
                        final HashMap params = new HashMap();
                        params.put("uname", accountParams.uname);
                        params.put("pass", accountParams.pass);
                        params.put("send", accountParams.send);
                        params.put("dest", normalizedPhoneNumber);
                        params.put("msg", text);
                        return
parseGatewayResponse(readRemoteData(buildUrl(accountParams.server, params)));
                }
                catch (final Exception ex)
                {
                        LOG.error("Error sending text[" + text + " ] to" +
normalizedPhoneNumber, ex);
                }
                return false;
        }

        @Override
        public void send(final AggregatorSettingsDTO settings, final
OutgoingSMSMessageDTO message) throws SMSEngineException
        {
                String phone;
                phone = message.getPhoneNumber();
                /*
                 * if (StringUtils.isNotEmpty(message.getNormalizedPhoneNumber()))
{ phone = message.getNormalizedPhoneNumber(); }
                 * else { final NormalizedPhoneNumber normalizedPhoneNumber =
                 *
this.phoneNumberService.validateAndNormalizePhoneNumber( message.getPhone
CountryIsoCode(),
                 * message.getPhoneNumber()); if (!
(normalizedPhoneNumber.isValid())) { throw new SMSEngineException( invalid
                 * number + message.getPhoneCountryIsoCode() + / +
```

```
message.getPhoneNumber() + : +
                * normalizedPhoneNumber.getErrorMessage()); }
                *
                * phone = normalizedPhoneNumber.getNormalizedNumber(); }
                */

               //       if (message.isLink()) {
               //              sendLink(phone, message.getSubject(),
message.getContent(),
               //                                    getAccountParams(settings));
               //} else {
               final boolean output = sendSms(phone, message.getContent(),
getAccountParams(settings));

               LOG.info("************Output of SMS Is::::::" + output);

               //              }
        }

        protected AccountParams getAccountParams(final AggregatorSettingsDTO
settings)
        {
               String url = settings.getAggregatorParameter("text.http.url");
               String uname = settings.getAggregatorParameter("text.http.uname");
               String pass = settings.getAggregatorParameter("text.http.pass");
               String sender = settings.getAggregatorParameter("text.http.send");

               if (StringUtils.isEmpty(url))
               {
                      url = Config.getParameter("text.http.url");
                      uname = Config.getParameter("text.http.uname");
                      pass = Config.getParameter("text.http.pass");
                      sender = Config.getParameter("text.http.send");
               }

               return new AccountParams(url, uname, pass, sender);
        }

        private static class AccountParams
        {
               final String server;
               final String uname;
               final String pass;
```

```
                final String send;

                AccountParams(final String server, final String uname, final String
pass, final String send)
                {
                        this.server = server;
                        this.uname = uname;
                        this.pass = pass;
                        this.send = send;
                }
        }
}
```

Config in <Your extension>facade-spring.xml

<Your extension>facade-spring.xml

```
<bean id= "mySMSEngine"
                class= "com.techouts.sms.engine.TestSendSMSEngine"
                parent= "abstractSMSEngine"
                scope= "tenant" >
        </bean>
```

Step 4:- Create an enumType in <YourExtention>Core-items.xml

<YourExtention>Core-items.xml

```
        <enumtypes>
        </enumtype>

<enumtype code= "EnginesType"  autocreate= "false"  generate= "false"  dynamic= "true" >
                        <value code= "mySMSEngine"  />
        </enumtype>
        </enumtypes>
```

NOTE: Make Sure that :

<enumtype code= EnginesType > will exactly match with the bean Id of SMS engine.

Step 5: Make a impex file to insert mobile Aggregrator and Short codes

```
INSERT_UPDATE MobileAggregator;  engine(code)  ;code[unique=true];
;  mySMSEngine  ;testOutgoingAggregator;



INSERT_UPDATE MobileShortcode;  aggregator(code)
;supportedMessageType(code);country(isocode);code[unique=true];&shortcode;
networkType(code)  ;
                        ;testOutgoingAggregator;OUTGOIN              ;  IN          ;
demoIN       ;demoES    ;GSM;
```

Step 6 : creating the itemtype for SendSMSProcess

<yourExtention>-items.xml

```
<typegroup>
            <itemtype code="SendSMSProcess"
extends="StoreFrontCustomerProcess"
                    autocreate="true" generate="true"

      jaloclass="com.techouts.core.jalo.process.sms.SendSMSProcess">
                    <description>Represents process that is used for Sending sms at
the time of registration .</description>
                    <attributes>
                        <attribute qualifier="firstName" type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>
                        <attribute qualifier="mobileNo" type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>
```

```
                        <attribute qualifier="template" type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>

                        <attribute qualifier="orderNumber"
type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>
                        <attribute qualifier="date" type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>

                        <attribute qualifier="orderStatus"
type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>

                        <attribute qualifier="delivryStatus"
type="java.lang.String">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>

                        <attribute qualifier="consignment" type="Consignment">
                            <modifiers read="true" write="true" search="true"
                                optional="true" />
                            <persistence type="property" />
                        </attribute>
                    </attributes>
                </itemtype>

        </typegroup>

        </itemtypes>
```

Step 7: Creating the Action class For sending SMS.

SendSMSAction.java

```
/**
 *
 */
package com.techouts.process.sms.actions;

import de.hybris.platform.core.model.c2l.CountryModel;
import de.hybris.platform.mobileservices.text.MobileActionParameters;
import de.hybris.platform.mobileservices.text.TextService;
import de.hybris.platform.processengine.action.AbstractProceduralAction;
import de.hybris.platform.servicelayer.i18n.CommonI18NService;
import de.hybris.platform.servicelayer.i18n.I18NService;
import de.hybris.platform.task.RetryLaterException;

import org.apache.log4j.Logger;

import com.techouts.core.model.process.sms.SendSMSProcessModel;


/**
 * @author jagadish
 *
 */
public class SendSMSAction extends
AbstractProceduralAction<SendSMSProcessModel>
{
	private TextService textService;
	private I18NService i18nService;
	private CommonI18NService commonI18NService;

	private static final Logger LOG = Logger.getLogger(SendSMSAction.class);

	@Override
	public void executeAction(final SendSMSProcessModel process) throws
RetryLaterException, Exception
```

```
	{

	LOG.info("**********************************************Now I
am in SendSMSAction Class by using process engine**********");
		try
		{
			final CountryModel country =
commonI18NService.getCountry("IN");
			final MobileActionParameters phone = new
MobileActionParameters();
			phone.setPhoneNumber(process.getMobileNo());
			phone.setCountryIsocode(country.getIsocode());

	phone.setLanguageIsocode(this.i18nService.getCurrentLocale().toString());

			this.textService.sendMessage(phone.getCountryIsocode(),
phone.getPhoneNumber(), process.getTemplate());


LOG.info("**********************************************************
** ");

		}
		catch (final Exception e)
		{
			LOG.error("error sending message to " + process.getMobileNo(),
e);
		}

	}

	/**
	 * @return the textService
	 */
	public TextService getTextService()
	{
		return textService;
	}

	/**
	 * @param textService
	 *           the textService to set
	 */
```

```
	public void setTextService(final TextService textService)
	{
		this.textService = textService;
	}

	/**
	 * @return the i18nService
	 */
	public I18NService getI18nService()
	{
		return i18nService;
	}

	/**
	 * @param i18nService
	 *           the i18nService to set
	 */
	public void setI18nService(final I18NService i18nService)
	{
		this.i18nService = i18nService;
	}

	/**
	 * @return the commonI18NService
	 */
	public CommonI18NService getCommonI18NService()
	{
		return commonI18NService;
	}

	/**
	 * @param commonI18NService
	 *           the commonI18NService to set
	 */
	public void setCommonI18NService(final CommonI18NService
commonI18NService)
	{
		this.commonI18NService = commonI18NService;
	}


}
```

Configure in facade spring.xml:

<YourExtetntion>Facade-Spring.xml

```
<bean id="sendSMS" class="com.techouts.process.sms.actions.SendSMSAction"
            parent="abstractAction">
                    <property name="textService" ref="textService" />
              <property name="i18nService" ref="i18nService"/>
              <property name="commonI18NService" ref="commonI18NService"/>
</bean>
```

Create GenerateSMSAction class to generate the action.

GenerateSMSAction.java

```
/**
 *
 */
package com.techouts.process.sms.actions;

import de.hybris.platform.core.model.user.CustomerModel;
import de.hybris.platform.processengine.action.AbstractProceduralAction;
import de.hybris.platform.task.RetryLaterException;

import org.apache.log4j.Logger;

import com.techouts.core.model.process.sms.SendSMSProcessModel;


/**
 * @author jagadish
```

```
 *
 */
public class GenerateSMSAction extends
AbstractProceduralAction<SendSMSProcessModel>
{
	private static final Logger LOG = Logger.getLogger(SendSMSAction.class);

	/*
	 * (non-Javadoc)
	 *
	 * @see
	 *
de.hybris.platform.processengine.action.AbstractProceduralAction#executeAction(de.hybris.platform.processengine
	 * .model.BusinessProcessModel)
	 */
	@Override
	public void executeAction(final SendSMSProcessModel process) throws
RetryLaterException, Exception
	{

	LOG.info("**************************************************Now I
am in SendSMSAction Class by using process engine**************");
		final CustomerModel customerModel = process.getCustomer();



	LOG.info("**************************************************Name
of the Customer Is : " + customerModel.getName());


LOG.info("**************************************************Mobile
Number of the Customer Is : "
				+ customerModel.getMobileNumber());
		//process.setFirstName(customerModel.getName());
		final String template = "Hi " + process.getFirstName() + " , Thank you
for Registering with V2kart.";



LOG.info("**************************************************Template
IS : " + template);

		process.setMobileNo(customerModel.getMobileNumber());
```

```
                process.setTemplate(template);
                getModelService().save(process);
        }

        //      final String orderNo = "123456789";
        //              final DateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        //              final Date date = new Date();
        //
        //              final String template = "Dear " + registerData.getFirstName() + "
thanks for placing order no " + orderNo
        //                              + ". EST DELIVERY DATE " +
dateFormat.format(date) + ". Check mail for details.";
}
```

Configure in <Extension>-facade-spring.xmll

```
        <bean id="generateSMS"
class="com.techouts.process.sms.actions.GenerateSMSAction"
                parent="abstractAction">
        </bean>
```

Step 8:Write one process in <YourExtetntion>.process

<YourExtetntion>.process

```
<?xml version="1.0" encoding="utf-8"?>
<process xmlns="http://www.hybris.de/xsd/processdefinition" start="generateSMS"
name="customerRegistrationSMSProcess"

        processClass="com.techouts.core.model.process.sms.SendSMSProcessModel"
onError="error">

        <action id="generateSMS" bean="generateSMS">
                <transition name="OK" to="sendSMS"/>
                <transition name="NOK" to="error"/>
```

```
        </action>

        <action id="sendSMS" bean="sendSMS">
                <transition name="OK" to="success"/>
                <transition name="NOK" to="failed"/>
        </action>

        <end id="error" state="ERROR">Something went wrong.</end>
        <end id="failed" state="FAILED">Could not send customer registration
sms.</end>
        <end id="success" state="SUCCEEDED">Sent customer registration
sms.</end>

</process>
```

Configure the process In <Your Extension>core-spring.xml:

<Your Extension>core-spring.xml:

```
<!--    Added for sms send process -->
        <bean id="Customerregistersendsmsprocess"


class="de.hybris.platform.processengine.definition.ProcessDefinitionResource">
                <property name="resource"


value="classpath:/lycamobilecore/processes/customerRegistrationSMSProcess.xml"
/>
        </bean>
```

All the things are created. for the example we have to call the Process From where we want.for example

LycacustomerRegisteFacede.java

```
public class LycacustomerRegisteFacede extends DefaultCustomerFacade
{
	@Autowired
	private LycaMobileCustomerReversePopulater lycacustomerpopulater;
	@Resource
	private ModelService modelService;
	@Resource
	private BusinessProcessService businessProcessService;

	@Override
	public void register(final RegisterData registerData) throws
DuplicateUidException
	{
		final Logger log =
Logger.getLogger("LycaMobileCustomerReversePopulater.class");
		validateParameterNotNullStandardMessage("registerData",
registerData);
		Assert.hasText(registerData.getFirstName(), "The field [FirstName]
cannot be empty");
		Assert.hasText(registerData.getLastName(), "The field [LastName]
cannot be empty");
		Assert.hasText(registerData.getLogin(), "The field [Login] cannot be
empty");

		final CustomerModel newCustomer =
getModelService().create(CustomerModel.class);


newCustomer.setName(getCustomerNameStrategy().getName(registerData.getFirst
Name(), registerData.getLastName()));

		if (StringUtils.isNotBlank(registerData.getFirstName()) &&
StringUtils.isNotBlank(registerData.getLastName()))
		{


newCustomer.setName(getCustomerNameStrategy().getName(registerData.getFirst
Name(), registerData.getLastName()));
		}
		final TitleModel title =
```

```
getUserService().getTitleForCode(registerData.getTitleCode());
                newCustomer.setTitle(title);
                log.info("calling mobile number method");
                //newCustomer.setMobileNumber(registerData.getMobileNumber());

                lycacustomerpopulater.populate(registerData, newCustomer);
                newCustomer.setMobileNumber(registerData.getMobileNumber());
                setUidForRegister(registerData, newCustomer);


newCustomer.setSessionLanguage(getCommonI18NService().getCurrentLanguage()
);


newCustomer.setSessionCurrency(getCommonI18NService().getCurrentCurrency())
;
                getCustomerAccountService().register(newCustomer,
registerData.getPassword());
                smsSending(newCustomer, registerData.getFirstName());
        }


 public void smsSending(final CustomerModel customer, final String
firstName)
        {
                final SendSMSProcessModel process =
businessProcessService.createProcess(String.valueOf(System.currentTimeMilli
s()),
                                "customerRegistrationSMSProcess");
                process.setCustomer(customer);
                process.setFirstName(firstName);
                modelService.save(process);
                businessProcessService.startProcess(process);
        }
}|
```