Document Purpose

How to write a delivery connector in Javascript, SOAP and as a workflow activity

Writing custom SMS connectors is not supported and highly discouraged.

For all SMS integrations past year 2018, you should use the Extended generic SMPP connector with ACC or the ACS connector. The extended generic SMPP connector and the ACS connector are the only ones supported by R&D.

These 2 connectors are described in the following page and its links: ACS SMPP connector and ACC extended generic SMPP connector [Out of date]

Overview

In version 4.05 (starting with build 1872), some delivery types (SMS, FAX and Paper) can now be executed internally, instead of generating export files. This brings the following benefits:

- access to the personalization engine for the message content
- parallelization of the delivery process (each delivery is spitted in independent parts), which also brings near-real time delivery capabilities for these channels
- automatic management of the delivery retries in case of technical errors
- capability to integrate with an existing delivery infrastructure

For this, a delivery connector must be created. The following delivery connectors are already built:

- NetSize connector, to handle the mobile channel (SMS, WAP Push and MMS)
- Maileva connector, to handle the direct mail delivery

Limitations

- this is not available for the Email channel, which remains a core functionality of the product.

What is a connector

A connector implements three methods:

- processDeliveryPart: to send a list of messages.
- getStatus: to asynchronously get the status of messages previously sent.
- getMessages: to get a list of incomming messages (optional, for the SMS channel).

The are three different possible implementations for a connector:

- as an external WebService, that Neolane calls through SOAP
- as a JavaScript package implemented within Neolane itself
- as a workflow woken up during the delivery process (but note that this type of implementation may be removed from subsequent versions, so do not use it).

Here is a matrix to help decide which implementation is best suited for your case:

Implementation	Benefits	Drawbacks
External WebService	• can be implemented in what ever language you want (Java, C#, PHP...) • easier to call an external API • can be implemented in Java and executed right within the Neolane application server (exposed as a JSP for instance)	• more complex to write

Internal JavaScript	• easy to write • no need for an external developement environnement	• can interact with the outside world with only what is exposed in our JavaScript implementation: files, HTTP, SOAP Calls, command lines • cannot wall a third party API exposed as a JAR file or as dynamic library (.dll, .so)

Methods

processDeliveryPart method

This methods takes a single input argument which is an XML document and is expected to output another XML document. It is called each time a nlserver mtachild process has personnalized a delivery fragment and need to process it through the external delivery connector. So, a single delivery can lead to numerous calls to this method. Also note that this method can be called simultaneously from multiple nlserver processes, so it needs to be reentrant.

The overall structure of the input document is as follows:

```
 <deliveryPart delivery-id="(number)" id="(number)" rcpCount="(number)" retry="(number)" validityDate="
(date)" lang="(text)">
   <message id="(number)" address="(text)" .../>
   ...
   <message id="(number)" address="(text)" .../>
   <options ... />
   <login account="(text)" password="(clear text)" port="(number)" server="(text)">
     <params ... />
   </login>
 </deliveryPart>
```

Notes:

- all dates are expressed using the ISO format 'YYYY/MM/DD HH:MM:SS'
- lang is the language in which error and logs messages should be produced in the output document. If you do not handle this, please output messages in english.
- the top-level delivery-id and id are more for information purposes (the represent the internal identifiers of the delivery action and the delivery part being processed)
- validityDate is the expiration date for the messages of the delivery fragment. Past this date, the messages should not be sent.
- the login element contains generic connexion parameters and a sub-element with connector-specific parameters, as defined in the nms:externalAccount schema.
- each message element has a custom sub-element containing the various profile values which have been fetched in order to create the personalized message.

This method must return an XML document which uses the following structure:

```
 <messages successOnSent="(boolean)">
   <message id="(number)"
            status="(number)"
            noRetry="(boolean)"
            log="(text)"
            warning="(text)"
            failureReason="(number)"
            failureType="(number)"/>
 </messages>
```

Notes:

Attribute	Description	Default value
successOn Sent	Indicates if the message should be considered sent in case of success. If false, then consider by default that the message has been forwarded to the connector. So this parameter has an impact on the default value of status (see below).	false
id	Identifier of the message given in the input document.	
providerId	The identifier of the message as returned by the SMSC. If several SMS's have been sent for a single message, return a list of comma sepeated Ids (with nothing else than the Ids and the commas, no spaces or anything like that).	

status	The processing status of the message. Possibles values are: • 0: DELIVERYSTATUS_IGNORED • 1: DELIVERYSTATUS_SENT • 2: DELIVERYSTATUS_FAILED • 3: DELIVERYSTATUS_SENDPENDING • 5: DELIVERYSTATUS_RECEIVED • 6: DELIVERYSTATUS_TOSEND • 7: DELIVERYSTATUS_NOTSENT • 8: DELIVERYSTATUS_PREPARED • 9: DELIVERYSTATUS_HANDEDOVER	9 (DELIVERYSTATUS_HANDEDOVER) if successOnSent is false, 1 (DELIVERYSTATUS_SENT) is successOnSent is true.
noRetry	If true, indicates that the delivery of this message should not be retried by neolane, even if the status indicates an error.	false
log	The text whihc should be used to update the NmsBroadLog table for this message. Please enforce the lang attribute of the deliveryPart.	Generic sucess/error message deduced form the status
warning	If used instead of log, the message log will be displayed as a warning in neolane. Applies only if the status is not an error. For instance, can be used in the sms channel to notify that a message was cut since too long.	
failureReason	In case of error (status=2), the reason of the error. Possibles values are: • 1: FAILUREREASON_UNKNOWN_USER • 2: FAILUREREASON_INVALID_DOMAIN • 3: FAILUREREASON_UNREACHABLE • 4: FAILUREREASON_DISABLED • 5: FAILUREREASON_MAILBOX_FULL • 6: FAILUREREASON_NOT_CONNECTED • 20: FAILUREREASON_REFUSED • 25: FAILUREREASON_ERRORIGNORED	3 (FAILUREREASON_UNREACHABLE)
failureType	In case of error (status=2), how to deal with this error regarding to quarantines. Possibles values are: • 1: FAILURETYPE_SOFT: this error increase the error counter and if a threshold of successive errors is reached, the adress is put in quarantine. • 2: FAILURETYPE_HARD: this error triggers an immediate quarantine. • 3: FAILURETYPE_IGNORED: this error is ignored in the quarantine mechanism.	3 (FAILURETYPE_IGNORED)

Notes:

- if for a message in the output document, no message element is present in the output document, the message is considered as sent.
- be careful with the noRetry value: if there is an error but there is a chance that the message has been sent anyway, please set it to true, otherwise, neolane will retry it and the message will be send multiple times (very annoying for SMS for instance).

Case of a mobile connector

The deliveryPart element has an additionnal mobileType attribute which indicates the type of mobile delivery.

```
<deliveryPart ... mobileType="(number)">
  <options class="(integer)" priority="(integer)"/>
  ...
</deliveryPart>
```

Attribute	Description	Default value
mobileType	• 0: SMS • 1: WAP Push • 2: MMS	0 (SMS)
class	• 0: immediate • 1: simspecific • 2: mobilespecific • 3: terminalspecific	1 (simspecific)
priority	• 0: normal • 1: high • 2: urgent	0 (normal)

For an SMS delivery, each message element is like:

```
<message address="(phone number)"  id="(number)">
  <text><![CDATA[(text)]]></text>
</message>
```

For a WAP Push delivery, each message element is like:

```
<message address="(phone number)" id="(number)"
         ulr="(text)">
  <text><![CDATA[(text)]]></text>
</message>
```

For a MMS delivery, each message element is like:

```
<message address="(phone number)" id="(number) subject="(text)">
  <mms>
    <text id="(text)"><![CDATA[...]]></text>
    ...
    <smil>
      xxx
    </smil>
  </mms>
</message>
```

The method means sending Mobile Terminated (MT) messages.

Case of a FAX connector (to be completed)

The deliveryPart element has an additionnal mobileType attribute which indicates the type of mobile delivery.

```
<deliveryPart ... pdfType="(number)" pdfCompression="(boolean)">
  <fileList>
    ...
  </fileList>
  ...
</deliveryPart>
```

Each message element is like:

```
<message address="(phone number)"  id="(number)">
  <faxText><![CDATA[(text)]]></faxText>
  <pdfFile id="()" fileName="(text)" startPage="(number)"/>
</message>
```

Notes:

- if pdfCompression is true, multiple messages are generated in the same pdf (different pages), otherwise there is one pdf per message.

Case of a Postal Mail (paper) connector (to be completed)

The deliveryPart element has an additionnal mobileType attribute which indicates the type of mobile delivery.

```
<deliveryPart ... pdfType="(number)" pdfCompression="(boolean)">
  <paperParameters>
  </paperParameters>
  <fileList>
    ...
  </fileList>
  ...
</deliveryPart>
```

getStatus method

This method takes a single input argument which is an XML document and is expected to output another XML document. It is called on a regular basis by neolane to get the delivery status of messages previously sent to the connector by the processDeliveryPart and for which the status was 9 (DELIVERYSTATUS_HANDEDOVER). For the mobile connector, it means getting Status Request (SR) messages.

The overall structure of the input document is as follows:

```
<getStatus maxMessages="(number)">
  <login account="(text)" password="(clear text)" port="(number)" server="(text)">
    <params ... />
  </login>
</getStatus>
```

This method must return an XML document which uses the same structure as the return document of the processDeliveryPart document. In addition, an error element can be returned to carry local non blocking errors (like impossible to get SMS status, but ok for MMS).

Notes:

- lang is the language in which error and logs messages should be produced in the output document. If you do not handle this, please output messages in english.
- the login element contains generic connexion parameters and a sub-element with connector-specific parameters, as defined in the nms:externalAccount schema.
- as of build 1883, it is implemented only for the mobile channel
- it is the connector's responsibility to return the identifier of messages as given during the processDeliveryPart.

If the API you are using in processDeliveryPart gives you back an id of it's own for each message pushed, you will have the manage the associative table yourself.

- the connector must return the delivery status of all message types (SMS, WAP Push, MMS) when called with this method. So if three calls are necessary internally, it is the connector responsibility to call them. If one of sub-calls fails, the overall call must not fail, and an error element can be stored in the returned document to send back this local error.

getMessages method (mobile channel only)

This method takes a single input argument which is an XML document and is expected to output another XML document. It applies only to the mobile channel and is called by the incomming sms activity in the workflows to get new inbound sms (Mobile Originated - MO).

The overall structure of the input document is as follows:

```
<getStatus maxMessages="(number)">
  <login account="(text)" password="(clear text)" port="(number)" server="(text)">
    <params ... />
  </login>
</getStatus>
```

This method must return an XML document which uses the following structure:

```
<messages>
  <message message="(text)"
           alias="(text)"
           separator="(text)"
           origin="(text)"
           messageId="(number)"
           deliveryDate="(date)"
           receivalDate="(date)"
           messageDate="(date)"
           largeAccount="(text)"
           countryCode="(number)"
           operatorCode="(number)"/>
</messages>
```

Notes:

- repeat the message element up to maxMessages times.
- each returned message element must specify at least message, origin and messageDate
- message: the text of the short message without alias and separator (if present) "MSFT"
- separator: separator found in message (if alias present) " "
- the alias found in message (if present) "B"
- origin: the MSISDN of the sender "+33612345678"
- alias: the alias found in message (if present) "B"
- deliveryDate: date of delivery indicated by the SMS-C
- receivalDate: date when the NSO driver received the message
- messageDate: date when the message is inserted into the database of the gateway
- messageId: the NSO MO ticket identifier
- largeAccount: key account number of the operator from which the message has been received "20100"
- countryCode: mobile Country Code of the operator to which the sender has subscribed
- operatorCode: mobile Network Code of the operator to which the sender has subscribed

When testing in-bound SMS in a workflow, there is a way to simulate the arrival of SMS, without actualy calling the connector. For this, add a debugMO parameter in the incoming event of the in SMS activity. Example:

```
vars.debugMO = <messages>
  <message message="coucou"
           alias="(text)"
           separator="(text)"
           origin="(text)"
           messageId="(number)"
           deliveryDate="(date)"
           receivalDate="(date)"
           messageDate="(date)"
           largeAccount="(text)"
           countryCode="(number)"
           operatorCode="(number)"/>
</messages>.toXMLString();
```

How to implement your connector

Create an external account

First, you need to create a new external account (in the Administration/External accounts folder), using the following attributes:

For a mobile connector

If you have specific connexion parameters (in addition to login/password/server/port) you may first have to extend the nms:extAccount schema and form to be able to store and edit them. Otherwise, stick with the Generic connector type.

Attribute	Value
Type	Routing
Message type	SMS
Delivery Mode	Bulk
Connector	Generic or your own new type after extending the schema.
Activation Mode	Call JavaScript or Call Web Service
JavaScript used in connector	When the activation mode is 'Call JavaScript', link to a JavaScript entity which implements the connector.
Access URL of the connector	When the activation mode is 'Call Web Service', HTTP Url which imlements the connector Soap methods (so XML over HTTP POST).

Implement the connector (JavaScript case)

Empty connector

Here is a sample JavaScript connector implementation. It does nothing, but allow to 'see' the actual data passed to the method calls:

```
function processDeliveryPart(xmlDeliveryPart)
{
  logInfo(xmlDeliveryPart.toXMLString());
  // Consider all ok
  return <messages/>
}

function getStatus(xmlRequest)
{
  logInfo(xmlRequest.toXMLString());
  // Consider no status available
  return <messages/>
}

function getMessages(xmlRequest)
{
  logInfo(xmlRequest.toXMLString());
  // Consider no messages available
  return <messages/>
}
```

Precautions

When implementing a method which iterate on messages, be careful of you interact using plain HTTP and use our **getUrl()** JavaScript function: this function throws an exception if the URL cannot be found, so your JavaScript function is aborted. You need to wrap the calls to getUrl in **try /catch** blocks, so that you can capture the errors and continue iterating on messages