SMS Alert

Whitepaper

Connection SMS Alert

VERSION 1.2, MAR 2010

Contents

+

1 Abbreviations and terms

API	Application Programming Interface. A collection of routines/functions that software applications use to carry out various operations.
IMSI	IMSI is an acronym for International Mobile Subscriber Identity. This is a unique number associated with the SIM card for GSM/UMTS.
LA	Abbreviation for "Large Account". Terminals/applications for sending and receiving messages that are linked to the SMSC. The LA must have an SMS Alert Large Account and SMS Alert agreement with Telenor.
Mobil terminal	The same as an MS or mobile phone.
MO-SM	Mobile Originated Short Message – message from an MS to the SMSC.
MSISDN	MS telephone number. In format "4799999999"
MS	Mobile Station, i.e. GSM or UMTS mobile terminal.
MT-SM	Mobile Terminated Short Message – message from the SMSC to an MS.
SMS	Short Message Service.
SMSC	Short Message Service Centre – message centre for SMS.
TCP/IP	Transmission Control Protocol / Internet Protocol.
VPN	Virtual Private Network.

2 General

SMS Alert can be used for population alert (disasters etc), corporate usage to send messages to employees within a specified area or country or in relation to content/marketing services.

SMS Alert is a service for the transmission of MT SM to GSM/3G terminals within a specified area. SMS Alert is limited to Telenor subscribers and MS roaming in Telenors Norwegian GSM/3G Network.

In short (some options are available):

- the SMS Alert customer defines a geographic area and the SMS content
- the SMS Alert customer uses the SMS Alert API to initiate the SMS Alert broadcast
- the SMS Alert system sends the SMS to end users in the specified area (includes end users who are within the coverage of the cells covering the specified area – that is, the SMS Alert broadcast will include some end users outside the specified area)
- the SMS Alert system sends a status/delivery report to the SMS Alert customer

Two way SMS can be achieved by combining SMS Alert with SMS Access.

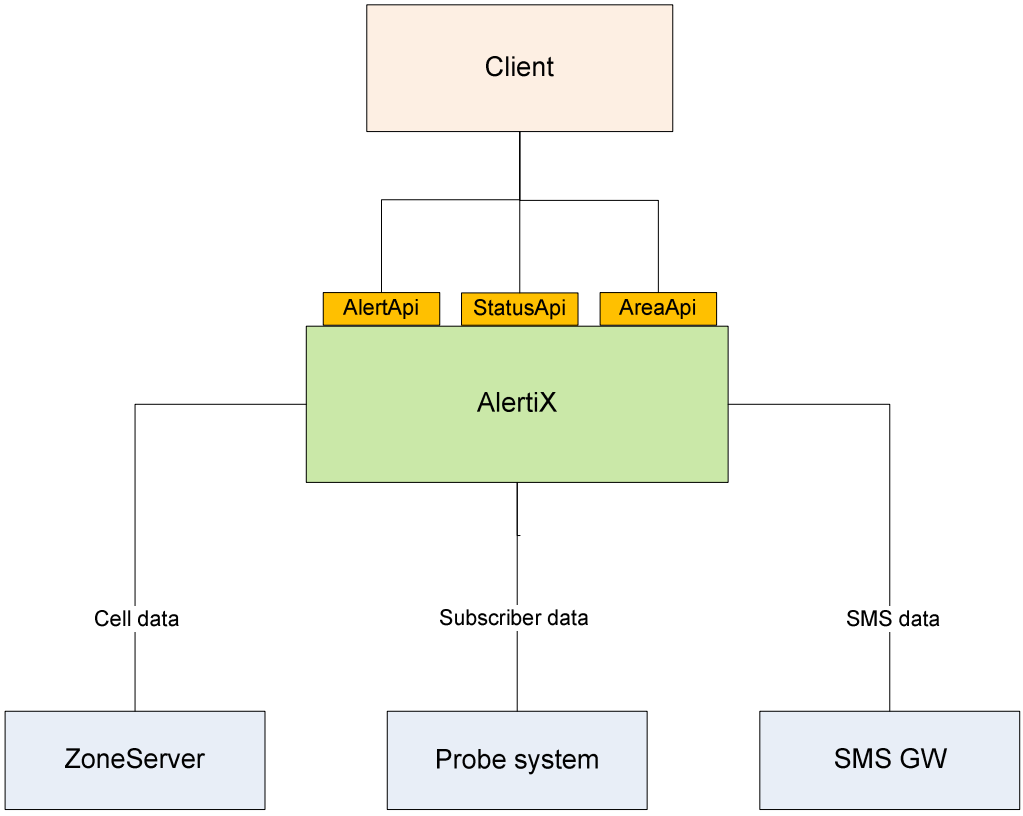
SMS Alert Large Accounts are configured as either the Full broadcast or Limited broadcast version.

SMS alert comes in two versions:

1. Full broadcast
    - Sends messages to all MS detected in the defined area
2. Limited broadcast (White list version)
    - Sends messages to all MS that are detected in the defined area AND are in the LA specific white list

Please refer to chapter 6 for limitations.

The SMS distribution is described in SMS Access Whitepaper and is based on Telenor SMSC.



3 SMS Alert Capacity

3.1 Capacity

The SMS Alert platform is configured with a SMSC capacity of 400 MT SM/second.

4 Service description

Please refer to chapter 2 for an introduction to the service description.

4.1 Services for LAs

The commercial terms for SMS Alert are found in the Telenor general terms as well as in the SMS Alert Agreement.

More information can be found at www.telenorfusion.no

4.2 Connection to SMS Alert

LAs can either use Nordic Connect, VPN or the Internet (https) for connection to SMS Alert.

4.2.1 Nordic Connect

A Company with Nordic Connect can use this as a connection to the SMSC. Nordic Connect offers a higher quality connection to SMS Alert than other connections. Telenor can provide more information on Nordic Connect.

4.2.2 VPN

A VPN consists of nodes in a public network, the Internet in this case, that communicates between themselves and use encryption technology so that the traffic between the nodes cannot be read by unauthorised users, in the same manner as if the nodes were connected via a private network. See the special white paper for this product for more information on Telenor VPN.

4.2.3 Internet

It is possible to access SMS Alert over the Internet (https).

4.3 Billing of messages

The customer of SMS Alert is billed based on the number of SMS submitted by the SMSC, **regardless** of whether the SMS is actually delivered to the receiving terminal. If the SMSC rejects a message it will not be billed.

5 SMS Alert API

5.1 End point URLs

API	URL
AlertApi	https://access.mobil.telenor.no/alertix/AlertApi
AreaApi	https://access.mobil.telenor.no/alertix/AreaApi
StatusApi	https://access.mobil.telenor.no/alertix/StatusApi
InternationalAlertApi	https://access.mobil.telenor.no/alertix/InternationalAlertApi

https (tcp/443)

WSDLs:

AlertApi	https://access.mobil.telenor.no/alertix/AlertApi?wsdl
AreaApi	https://access.mobil.telenor.no/alertix/AreaApi?wsdl
StatusApi	https://access.mobil.telenor.no/alertix/StatusApi?wsdl
InternationalAlertApi	https://access.mobil.telenor.no/alertix/InternationalAlertApi?wsdl
AdminApi	https://access.mobil.telenor.no/alertix/AdminApi?wsdl

5.2 Alert types

There are four different ways to start an alert. An alert can either be an area-type alert or a custom-type alert. Also an alert can either be a one-phase alert or a two-phase alert.

- Area Alert.
  The request specifies a predefined area name as the target of the alert. The area name and the associated polygon are defined using the AreaApi. In this case the cells to include in the alert are pre-calculated.
  Also, a list of MSISDN can be specified by the client. The alert will then cover a union of subscribers in the area and the subscribers in the MSISDN list.
- Custom Alert.
  The request specifies a polygon as the target of the alert. The cells to include in the alert are calculated on the fly during the alert operation.
  Also, a list of MSISDN can be specified by the client. The alert will then cover a union of subscribers in the area and the subscribers in the MSISDN list.
- International Alert.
  InternationalAlertApi is a Web Service API for starting population alert to domestic subscribers in foreign countries. The request specifies a list of country codes as the target of the alert. All subscribers in these countries will then be alerted.
  Also, a list of MSISDN can be specified by the client. The alert will then cover a union of subscribers in the area and the subscribers in the MSISDN list.
- One phase Alert.
  The alert is started with one command. The client must use the StatusApi to check the progression of the alert. There is no way to stop an alert after it is started.
- Two Phase Alert.
  The first phase is started with a prepare command which finds all subscribers and do all required processing, except submitting any messages
  Using the StatusApi the client can check the number of subscribers and the country code distribution.
  In the second phase the prepared alert is executed meaning that the messages are submitted.

The name of the alert methods is explained in the following table:

	API	One Phase Alert	Two Phase Alert
Area Alert	AlertApi	*executeAreaAlert*	*prepareAreaAlert* / *executePreparedAlert*
Custom Alert		*executeCustomAlert*	*prepareCustomAlert* / *executePreparedAlert*
International Alert	International AlertApi	*executeAlert*	*prepareAlert* / *executePreparedAlert*

5.2.1 Alert API description

```
interface AlertApi {

  AlertResponse executeAreaAlert (AreaName areaName, AlertMsg alertMsg,
                    WhiteLists whiteLists,
                    AdditionalSubscribers additionalSubscribers,
                    ExecuteMode executeMode)

  AlertResponse executeCustomAlert (AlertMsg alertMsg,
                    WhiteLists whiteLists,
                    Polygon areaPolygon,
                    AdditionalSubscribers additionalSubscribers,
                    ExecuteMode executeMode)

  AlertResponse prepareAreaAlert (AreaName areaName, AlertMsg alertMsg,
                    WhiteLists whiteLists,
                    AdditionalSubscribers additionalSubscribers)

  AlertResponse prepareCustomAlert (AlertMsg alertMsg,
                    WhiteLists whiteLists,
                    AdditionalSubscribers additionalSubscribers,
                    Polygon areaPolygon)

  AlertResponse executePreparedAlert (JobId jobId,
                    ExecuteMode executeMode)

  Response      cancelPreparedAlert (JobId jobId)

  AlertResponse executeAdditionalAlert(JobId initialJobId,
                   AlertMsg alertMsg, ExecuteMode executeMode)


 }
```

Information Element	Comment
areaName	A logical name of the area to alert.
alertMsg	An object specifying the message content and the expiry time of the message. This object contains a default message and country code specific messages. The default message applies when the subscribers country code does not match any of the country codes in this message.
whiteLists	An object which contains a list of white list names. Each white list name identifies a white list which contains a list of predefined MSISDNs. If one or more white lists is specified, then the alert will only be submitted to the subscribers in a union of the white lists. If the white lists parameter is null of the WhiteList object contains an empty list, the white list functionality is disabled.
additionalSubscribers	An object which contains a list of MSISDN, specifying subscribers to be alerted independent of the area specified. If this parameter is null or the list is empty, then only those subscribers found in the actual areas are alerted. The additionalSubscribers will typically be alerted after the subscribers from the actual areas are alerted.
executeMode	Specifies a mode which can be either LIVE or SIMULATE. If the mode is SIMULATE, no message is actually submitted to subscribers, and delivery reports are faked so that the StatusApi will report that most messages is delivered successfully and some are delivered with failure. If mode is LIVE all subscribers will be alerted.
areaPolygon	Specifies the geographical area as a set of (X,Y) points. The coordinates system

	used will depend on the operator.
JobId	A unique identifier of an alert.
Response	A response object which contains a result code and a description
AlertResponse	A response object which contains the JobId in addition to the result code and the description.

```
interface AlertMsg {
    TextMessage           getDefaultMessage();
    CountryTextMessage[]  getCountryMessages();
    dateTime              getExpiryTime ()
}
```

Information Element	Comment
getDefaultMessage	Default message if no country-specific message applies. This object contains both the text content and the originating address (the sender of the SMS)
getCountryMessages	Specifies a mapping between a collection of country codes and a text message/originating address-pair.
getExpiryTime	The expiry time is an absolute timestamp. The expiry time is used as a validation time for SMS messages i.e. SMS'es will not be delivered to end user terminals if this time is exceeded. The expiry time is also used with prepared jobs. A prepared job cannot be executed after the expiry time has expired.

```
interface CountryTextMessage extends TextMessage {
    int[]   getCountryCodes ();
}
```

Information Element	Comment
getText	The Text to submit to subscribers.
getOa	Originating address for the message.
getCountryCodes	This message should be applied to these country codes. If this array is empty, this message is assumed to be the default language.

5.2.2 InternationalAlertAPI description

```
interface InternationalAlertApi {

  AlertResponse executeAlert (CountryCodes countryCodes,
                    InternationalAlertMsg alertMsg,
                    WhiteLists whiteLists,
                    AdditionalSubscribers additionalSubscribers,
                    ExecuteMode executeMode)

  AlertResponse prepareAlert (CountryCodes countryCodes,
                    InternationalAlertMsg alertMsg,
                    WhiteLists whiteLists,
                    AdditionalSubscribers additionalSubscribers)

  Response executePreparedAlert (JobId jobId, ExecuteMode executeMode)

  Response cancelPreparedAlert (JobId jobId)

  AlertResponse executeAdditionalAlert(JobId initialJobId,
                    InternationalAlertMsg alertMsg,
                    WhiteLists whiteLists,
                    ExecuteMode executeMode)

 }
```

Information Element	Comment
countryCodes	An object which contains a list of country codes, defining which countries the targeted subscribers are located in.
alertMsg	An object specifying the message content and the expiry time of the message.
whiteLists	An object which contains a list of white list names. Each white list name identifies a white list which contains a list of predefined MSISDNs. If one or more white lists is specified, then the alert will only be submitted to the subscribers in a union of the white lists. If the WhiteLists parameter is null of the WhiteList object contains an empty list, the white list functionality is disabled.
additionalSubscribers	An object which contains a list of MSISDN, specifying subscribers to be alerted independent of the area specified. If this parameter is null or the list is empty, then only those subscribers found in the actual areas are alerted. The additionalSubscribers will typically be alerted after the subscribers from the actual areas are alerted.
executeMode	Specifies a mode which can be either LIVE or SIMULATE. If the mode is SIMULATE, no message is actually submitted to subscribers, and delivery reports are faked so that the StatusApi will report that most messages is delivered successfully and some are delivered with failure. If mode is LIVE all subscribers will be alerted.
JobId	A unique identifier of an alert.
Response	A response object which contains a result code and a description
AlertResponse	A response object which contains the JobId in addition to the result code and the description.

```
interface InternationalAlertMsg {
      TextMessage           getMessage();
      dateTime              getExpiryTime ()
  }
```

Information Element	Comment
getMessage	This object contains both the text content and the originating address (the sender of the SMS)
getExpiryTime	The expiry time is an absolute timestamp. The expiry time is used as a validation time for SMS messages i.e. SMS'es will not be delivered to end user terminals if this time is exceeded. The expiry time is also used with prepared jobs. A prepared job cannot be executed after the expiry time has expired.

```
interface TextMessage {
    String  getText ();
    String  getOa ();
}
```

Information Element	Comment
getText	The Text to submit to subscribers.
getOa	Originating address for the message.

Note: Only the "`executeAlert(...)`" method will be implemented at launch of the API.

5.2.3 Area API

Web Service API for configuring area definitions etc.

For Telenor Norway a simple map tool for retrieving the UTM33 coordinates can be found at: http://kart.statkart.no/adaptive2/default.aspx

And an overview of base stations in Norway can be found at: http://www.finnsenderen.no/search

```
interface AreaApi {
    Response        createArea (Polygon polygon, AreaName areaName);
    Response        deleteArea (AreaName areaName);
    Response        deleteAllAreas ();
    GetAreaResponse       getArea (AreaName areaName);
    GetAllResponse        getAllAreas ();
    void            updateArea (AreaName areaName, Polygon polygon);
}
```

Information Element	Comment
polygon	Specifies the geographical area as a set of (X,Y) points. The coordinates system used will depend on the operator.
areaName	A logical name of the area.
GetAreaResponse	A response object which contains information about an area. Specifies an area with name, areaId and polygon as shown below.
GetAllResponse	A response object which contains a list of areas as shown below.

```
interface Area {
    AreaId getAreaId();
    String getAreaName();
    Point[] getPolygon();
}
```

5.2.4 Status API

Web Service API for monitoring alert propagation and message distribution propagation.

```
interface StatusApi {

   CountryCodeAlertStatusResponse getAlertStatusByCountryCode (
                    JobId jobId);

   JobStatusResponse getJobStatus(JobId jobId);

   AllJobsResponse getAllJobs();

 }
```

Information Element	Comment
jobId	The job to check status for
CountryCodeAlertStatusResponse	Response object which in addition to result code/description contains an array of CountryCodeAlertStatus objects as shown below. The array contains one entry per country code.
JobStatusResponse	Response object which in addition to result code/description contains the JobStatus of a specified alert: PREPARING, PROCESSING_SUBSCRIBERS, NO_SUBSCRIBERS, PREPARED, PREPARED_LOCK, SUBMITTING, ERROR
AllJobsResponse	Response object which in addition to result code/description contains an array of JobInfo objects as shown below.

```
interface CountryCodeAlertStatus {

     int   getCountryCode ();
     int   getSubscribersCount ();
     int   getInitialized ();
     int   getQueued ();
     int   getSubmitted ();
     int   getSubmissionFailed ();
     int   getDeliveredSuccessfully ();
     int   getDeliveryExpired ();
     int   getDeliveryFailed ();
     int   getDeliveryUnknown ();
 }
```

Information Element	Comment
getCountryCode	Country code for the statistics in this CountryStatus-object.
getSubscribersCount	The total subscribers in the area with the given country code. This is the sum of the subsequent counters.
getInitialized	This is the initial state.
getQueued	The number of messages which are processed by AlertiX and queued for being submitted to SMS API.
getSubmittedCount	Submitted to SMS API.
getDeliveredCount	Delivered to end-user
getDeliveryExpiryCount	Delivery timeout. Terminal switched off?
getDeliveryFailureCount	Delivery failed due to other cause than expiry.

```
interface JobInfo{
  JobId getJobId();
  AreaName getAreaName();
  JobStatus getJobStatus();
  UserId getExecutor();
  Date getStarted();
  Date getExpiry();
  ExecuteMode getExecuteMode();
}
```

Information Element	Comment
getAreaName	Name of the area
getJobStatus	The status of the job: PREPARING, PROCESSING_SUBSCRIBERS, NO_SUBSCRIBERS, PREPARED, PREPARED_LOCK, SUBMITTING, ERROR
getExecutor	The user which started the Job. In case of two phase alert, the executor will be the user which invoked the prepared method. After executePreparedAlert is invoked, the executor will be updated with the user which invoked that method.
getStarted	The timestamp when the job was started.
getExpiry	The expiry time as specified in the AlertMsg object.
getExecuteMode	The ExecuteMode of the job. When using two phase alert, the ExecuteMode will be unknown until executePreparedAlert.

5.2.5 Admin API

Web Service API for administration of groups, users and white lists.

Numbers for whitelist must always start with country code 47.

A Group is typically an organization (e.g. Police or Road authorities). A group is an owner of one or more areas. Each group contains one or more users. This user is the authenticated user (HTTP basic authentication) accessing the Alertix API's. Each user has a password and a set of roles associated with it. The roles specify which API's the user is authorized to access.

```
interface AdminApi {

    AllUsersResponse getAllUsers();

    UserResponse getUser(UserName name);

    Response deleteUser(UserName name);

    Response addUser(User name);

    Response updateUser(User name);

    WhiteListsResponse getWhiteLists();

    Response createWhiteList(WhiteListName name);

    Response deleteWhiteList(WhiteListName name);

    SubscribersResponse getWhiteListSubscribers(WhiteListName name);

    DeleteResponse deleteWhiteListSubscribers(WhiteListName name,
     Msisdn[] numbers);

    AddResponse addWhiteListSubscribers(WhiteListName name,
     Msisdn[] numbers);
}
```

5.3 Result codes

Name	Code	Comment
SUCCESS	200	The operation was successful. If the response object contains other information, that information is expected to be present i.e. not null.
AX_JOB_NOT_READY	201	The alert processing has started but the requested subscriber information is not yet available.
AX_NO_SUBSCRIBERS	202	No subscriber is found to alert.
AX_OBJECT_NOT_FOUND	800	The requested area or job is not valid.
AX_ZS_OUT_OF_SYNC	801	An intern error indicating that AleritX is out of sync with ZoneServer.
AX_JOB_EXPIRED	803	The Job expired, possible reasons: • A prepared job has expired. • The expiryTime specified in an AlertMsg occurred in the past
AX_INVALID_ALERTMSG	804	The specified AlertMsg is invalid, possible reasons: • The defaultMessage is missing (null)

		• The expireTime is missing (null)
AX_OBJECT_ALREADY_EXISTS	807	In AdminApi: A user or a whitelist with that name already exist.
AX_WHITELIST_REQUIRED	808	A mandatory whitelist is missing in the alert.
AX_ILLEGAL_PARAMETER	810	Validation of an input parameter fails.
AX_SYSTEM_ERROR	899	Unspecified system error
SECURITY_ERROR	902	The user is not allowed to perform the operation, possible reason: • The area belongs to another group.

6 Limitations

6.1 Near real time applications

Even if SMS Alert is designed for delivering a near real time service, it is important to be aware of the fact that the SMS basic service is not designed for transmission of data in real-time and that the elapsed time from sending to receiving may vary significantly, even when the service is fully operational. When the SMSC sends to a phone within radio coverage, the message will normally be delivered on the first transmission attempt.

A retry attempt after 5 minutes will normally be made to transmit messages that fail. This means that messages, that do not reach the recipient within a few seconds, could be delayed even without any special problems in the network.

Special circumstances (also the type of situations where SMS Alert is intended) with unusual traffic volumes or network failure can cause blocking in the network.

6.2 Reliability of the system to reach "all" customers

The SMS Alert system utilises probes to collect data from the Network to update the location of the end users. Under optimal conditions the probes should update the location of the end user every time there is a network update, data or circuit switched data transfer to or from the end user or if the user enters a new Location Area in the Network.

Tests show that the probes are able to interpret well above 90% of the collected data. There is still a percentage of the data that is not interpreted; this can however be crucial in updating the location of the end user.

It is likely that there will be people that have moved from one area to another area without being updated in the SMS Alert system.

6.3 Capacity

General SMSC load can reduce the capacity (especially Christmas and New Years Eve). Also multiple SMS Alert jobs will influence the individual capacity of each SMS Alert customer.

Multiple users within one cell could influence the SMS throughput in the network. The SMS Alert platform has logic to reduce the chance of network throughput problems. The data rate for the transmission an SMS message over the radio interface is << 1 kbit/sec. A single GSM base station can normally handle between 4 and 8 simultaneous SMS transmissions.

7 References

1 **Telenor VPN Solution**

8 Document history

Version	Date	Comments
1.1	16 Feb 2010	Updated after integration trials, added Java examples
1.2	3 Mar 2010	Updated error codes

A. Java example with StatusAPI

The stubs and skeletons are generated by the IDE tool from the WSDL files.

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import cellvision.alertix.api.StatusApi;
import cellvision.alertix.api.StatusApiLocator;
import cellvision.alertix.api.StatusApiPortType;
import cellvision.alertix.api.vo.AllJobsResponse;
import cellvision.alertix.api.vo.CountryCodeAlertStatusResponse;
import cellvision.alertix.common.domain.JobInfo;
import cellvision.alertix.common.domain.api.CountryCodeAlertStatus;

public class Test_Status {

	private static final String StatusLocation =
	"https://access.mobil.telenor.no/alertix/StatusApi";
	private static final String userName = "Your username";
	private static final String passWord = "Your password";

	void getAllJobs(){

		System.out.println("getAllJobs()");
		StatusApi sa = new StatusApiLocator();
		try {
			StatusApiPortType sapt = sa.getStatusApiHttpPort(new
			java.net.URL(StatusLocation));
			org.apache.axis.client.Stub stub =
			(org.apache.axis.client.Stub)sapt;
			stub.setUsername(userName);
			stub.setPassword(passWord);
			AllJobsResponse ajr = sapt.getAllJobs();
			Integer gc = ajr.getCode();
			String gm = ajr.getMessage();
			boolean gs = ajr.getSuccessful();
			JobInfo[] gji = ajr.getJobInfos();
			int noe = gji.length;
			// Success
			System.out.println("Result: Code:" + gc + " Message:" + gm
			+ " Successful:" + gs + " Lenght:" + noe);
			// Unpack result
			if(gji != null)for ( int n=0; n<gji.length ; n++ ){

				System.out.println("[" + n + "] "
		+ " JobId:" + gji[n].getJobId().getValue()
		+ " Area Name:" + gji[n].getAreaName().getValue()
		+ " JobStatus:" + gji[n].getJobStatus().getValue()
		+ " ExecuteMode:" + gji[n].getExecuteMode().getValue()
		+ " Executor:" + gji[n].getExecutor().getValue()
		+ " Expiry:" + gji[n].getExpiry().getTime()
				);
			}
			if(gji != null)for ( int m=0; m<gji.length;m++){
				CountryCodeAlertStatusResponse ccasr =
		sapt.getAlertStatusByCountryCode(gji[m].getJobId());
				System.out.println("[" + m + "] "
```

```
                                    + " Code:" + ccasr.getCode()
                                    + " Message:" + ccasr.getMessage()
                        );
                        CountryCodeAlertStatus[] ccas =
                        ccasr.getCountryStatusCounts();
                        if(ccas != null)
            for ( int mm = 0; mm<ccas.length; mm++){
                                System.out.println("[" + mm + "] "
            + " Country Code:" + ccas[mm].getCountryCode()
            + " Subscriber count:" + ccas[mm].getSubscribersCount()
            + " Expiry count:" + ccas[mm].getDeliveryExpired()
            + " Fail count:" + ccas[mm].getDeliveryFailed()
            + " Unknown count:" + ccas[mm].getDeliveryUnknown()
                                );
                        }
                }
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        /**
         * @param args
         */
        public static void main(String[] args) {
            System.out.println("Start:");
            Test_Status gs = new Test_Status();
            gs.getAllJobs();
        }
}
```

B. Java example with AdminAPI

The stubs and skeletons are generated by the IDE tool from the WSDL files.

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import cellvision.alertix.api.AdminApi;
import cellvision.alertix.api.AdminApiLocator;
import cellvision.alertix.api.AdminApiPortType;
import cellvision.alertix.api.vo.AddResponse;
import cellvision.alertix.api.vo.AllUsersResponse;
import cellvision.alertix.api.vo.SubscribersResponse;
import cellvision.alertix.api.vo.WhiteListsResponse;
import cellvision.alertix.common.domain.api.Msisdn;
import cellvision.alertix.common.domain.api.Password;
import cellvision.alertix.common.domain.api.Role;
import cellvision.alertix.common.domain.api.RoleList;
import cellvision.alertix.common.domain.api.User;
import cellvision.alertix.common.domain.api.UserName;
import cellvision.alertix.common.domain.api.WhiteListName;
import cellvision.alertix.common.vo.Response;

public class Test_Admin {
      private static final String AdminLocation =
      "https://access.mobil.telenor.no/alertix/AdminApi";
      private static final String userName = "Your username";
      private static final String passWord = "Your password";

      void getAllUsers(){

            System.out.println("getAllUsers()");
            AdminApi aa = new AdminApiLocator();
            try {
                  AdminApiPortType aapt = aa.getAdminApiHttpPort
                  (new java.net.URL(AdminLocation));
                  org.apache.axis.client.Stub stub =
                  (org.apache.axis.client.Stub)aapt;
                  stub.setUsername(userName);
                  stub.setPassword(passWord);
                  AllUsersResponse aur = aapt.getAllUsers();
                  Integer gc = aur.getCode();
                  String gm = aur.getMessage();
                  boolean gs = aur.getSuccessful();
                  User[] gu = aur.getUsers();
                  int noe = gu.length;
                  // Successful
                  System.out.println("Result: Code:" + gc + " Message:" + gm
                  + " Successful:" + gs + " Lenght:" + noe);
                  // Unpack result
                  if(gu != null)for ( int n=0; n<noe;n++){
                        System.out.println("[" + n + "] Username:" +
                        gu[n].getUserName().getValue() + " Password:" +
                        gu[n].getPassword().getValue());
                        RoleList rl = gu[n].getRoleList();
                        Role[] gr = rl.getRoles();
                        System.out.println("With roles:");
```

```
                        for ( int m = 0; m < gr.length; m++){
                              System.out.println("[" + m + "] " +
                              gr[m].getValue());
                        }
                  }
                  WhiteListsResponse gwl = aapt.getWhiteLists();
                  gc = gwl.getCode();
                  gm = gwl.getMessage();
                  gs = gwl.getSuccessful();
                  WhiteListName[] wln = gwl.getWhiteLists();
                  noe = wln.length;
                  // Successful
                  System.out.println("WhiteLists: " + noe);
                  // Unpack result
                  if(wln != null)for ( int nn=0; nn<noe ; nn++){
                        SubscribersResponse sr =
                        aapt.getWhiteListSubscribers(wln[nn]);
                        gc = sr.getCode();
                        gm = sr.getMessage();
                        gs = sr.getSuccessful();
                        Msisdn[] ms = sr.getMsisdns();
                        int noei = ms.length;
                        System.out.println("[" + nn + "] WhiteListName:"
                        + wln[nn].getValue() + " Code:" + gc
                        + " Message:" + gm + " Successful:" + gs
                        + " Lenght:" + noe);
                        System.out.print("Contains: ");
                        for (int m = 0; m < noei; m++){
                              System.out.print(" " + ms[m].getValue());
                        }
                        System.out.println();
                  }
            } catch (RemoteException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (MalformedURLException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            } catch (ServiceException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }

      private void deleteWhiteList(WhiteListName wln) {

            System.out.println("deleteWhiteList(WhiteListName wln) " +
            wln.getValue());
            AdminApi aa = new AdminApiLocator();
            try {
                  AdminApiPortType aapt = aa.getAdminApiHttpPort(new
                  java.net.URL(AdminLocation));
                  org.apache.axis.client.Stub stub =
                  (org.apache.axis.client.Stub)aapt;
                  stub.setUsername(userName);
                  stub.setPassword(passWord);
                  Response res = aapt.deleteWhiteList(wln);
                  Integer gc = res.getCode();
                  String gm = res.getMessage();
```

```
                boolean gs = res.getSuccessful();
                // Success
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs );
          } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          }
    }

    private void deleteUser(UserName usn) {

          System.out.println("deleteUser(UserName usn) : " + usn.getValue());
          AdminApi aa = new AdminApiLocator();
          try {
                AdminApiPortType aapt = aa.getAdminApiHttpPort(new
                java.net.URL(AdminLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)aapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response ar = aapt.deleteUser(usn);
                Integer gc = ar.getCode();
                String gm = ar.getMessage();
                boolean gs = ar.getSuccessful();
                // Success
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs  );
          } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
          }
    }
    private void addUser(User usr) {

          System.out.println("addUser(User usr) : " +
          usr.getUserName().getValue());
          AdminApi aa = new AdminApiLocator();
          try {
                AdminApiPortType aapt = aa.getAdminApiHttpPort(new
                java.net.URL(AdminLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)aapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response ar = aapt.addUser(usr);
                Integer gc = ar.getCode();
```

```
                String gm = ar.getMessage();
                boolean gs = ar.getSuccessful();
                // Success
                System.out.println("Result: Code:" + gc + " Message:" + gm + "
Successful:" + gs  );
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
      }

      private void addWhiteListMembers(WhiteListName wln, Msisdn[] ms) {

            System.out.println("addWhiteListMembers(WhiteListName wln,
            Msisdn[] ms) : " + wln.getValue());
            AdminApi aa = new AdminApiLocator();
            try {
                AdminApiPortType aapt = aa.getAdminApiHttpPort(new
                java.net.URL(AdminLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)aapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                AddResponse ar = aapt.addWhiteListSubscribers(wln, ms);
                Integer gc = ar.getCode();
                String gm = ar.getMessage();
                boolean gs = ar.getSuccessful();
                Integer cnt = ar.getAddedCount();
                // Success
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs  + " Count:" + cnt);
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
      }

      private void createWhiteList(WhiteListName wln) {

            System.out.println("createWhiteList(WhiteListName wln) : " +
            wln.getValue());
            AdminApi aa = new AdminApiLocator();
            try {
                AdminApiPortType aapt = aa.getAdminApiHttpPort(new
                java.net.URL(AdminLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)aapt;
```

```
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response res = aapt.createWhiteList(wln);
                Integer gc = res.getCode();
                String gm = res.getMessage();
                boolean gs = res.getSuccessful();
                // Success
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs );
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        /**
         * @param args
         */
        public static void main(String[] args) {
            Test_Admin ga = new Test_Admin();
            ga.getAllUsers();
            WhiteListName wln = new WhiteListName();
            wln.setValue("test_list");
            ga.deleteWhiteList(wln);
            ga.createWhiteList(wln);
            Msisdn[] ms = new Msisdn[1];
            ms[0] = new Msisdn();
            ms[0].setValue("4799999999");
            ga.addWhiteListMembers(wln, ms);
            User usr = new User();
            UserName usn = new UserName("yourDemo");
            usr.setUserName(usn);
            usr.setPassword(new Password("yourDemo"));
            Role[] ra = new Role[3];
            ra[0] = Role.STATUS;
            ra[1] = Role.AREA;
            ra[2] = Role.ALERT;
            RoleList rl = new RoleList(ra);
            usr.setRoleList(rl);
            ga.deleteUser(usn);
            ga.addUser(usr);
        }
}
```

C. Java example with AreaAPI

The stubs and skeletons are generated by the IDE tool from the WSDL files.

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import cellvision.alertix.api.AreaApi;
import cellvision.alertix.api.AreaApiLocator;
import cellvision.alertix.api.AreaApiPortType;
import cellvision.alertix.api.vo.GetAllResponse;
import cellvision.alertix.common.domain.api.Area;
import cellvision.alertix.common.domain.api.AreaName;
import cellvision.alertix.common.domain.api.Point;
import cellvision.alertix.common.domain.api.Polygon;
import cellvision.alertix.common.vo.Response;

public class Test_Area {

	private static final String AreaLocation =
	"https://access.mobil.telenor.no/alertix/AreaApi";
	private static final String userName = "Your username";
	private static final String passWord = "Your password";

	void getAllAreas(){
		System.out.println("GetAllAreas()");
		AreaApi sa = new AreaApiLocator();
		try {
			AreaApiPortType sapt = sa.getAreaApiHttpPort(new
			java.net.URL(AreaLocation));
			org.apache.axis.client.Stub stub =
			(org.apache.axis.client.Stub)sapt;
			stub.setUsername(userName);
			stub.setPassword(passWord);
			GetAllResponse gar = sapt.getAllAreas();
			Integer gc = gar.getCode();
			String gm = gar.getMessage();
			boolean gs = gar.getSuccessful();
			Area[] a = gar.getAreas();
			String answer = null;
			if(a != null)answer = "" + a.length;
			// Successful
			System.out.println("Result: Code:" + gc + " Message:" + gm
			+ " Successful:" + gs + " Lenght:" + answer);
			if(a != null)for ( int n=0; n<a.length;n++){

				System.out.println("[" + n + "] " +
				a[n].getAreaId().getValue() + " " +
				a[n].getAreaName().getValue());
			}
		} catch (ServiceException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (RemoteException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (MalformedURLException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        private void deleteAllAreas() {

            System.out.println("deleteAllAreas()");
            AreaApi sa = new AreaApiLocator();
            try {
                AreaApiPortType sapt = sa.getAreaApiHttpPort(new
                java.net.URL(AreaLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)sapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response gar = sapt.deleteAllAreas();
                Integer gc = gar.getCode();
                String gm = gar.getMessage();
                boolean gs = gar.getSuccessful();
                // Successful
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs );
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        private void deleteArea(AreaName name) {

            System.out.println("deleteArea(AreaName name) : "
            + name.getValue());
            AreaApi sa = new AreaApiLocator();
            try {
                AreaApiPortType sapt = sa.getAreaApiHttpPort(new
                java.net.URL(AreaLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)sapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response gar = sapt.deleteArea(name);
                Integer gc = gar.getCode();
                String gm = gar.getMessage();
                boolean gs = gar.getSuccessful();
                // Successful
                System.out.println("Result: Code:" + gc + " Message:" + gm
                + " Successful:" + gs );
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
```

```
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        private void createArea(Polygon poly, AreaName name) {

            System.out.println("createArea(Polygon poly, AreaName name) : "
            + name.getValue());

            AreaApi sa = new AreaApiLocator();
            try {
                AreaApiPortType sapt = sa.getAreaApiHttpPort(new
                java.net.URL(AreaLocation));
                org.apache.axis.client.Stub stub =
                (org.apache.axis.client.Stub)sapt;
                stub.setUsername(userName);
                stub.setPassword(passWord);
                Response gar = sapt.createArea(poly, name);
                Integer gc = gar.getCode();
                String gm = gar.getMessage();
                boolean gs = gar.getSuccessful();
                // Successful
                System.out.println("Result: Code:" + gc + " Message:" + gm + "
Successful:" + gs );
            } catch (ServiceException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (RemoteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (MalformedURLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        /**
         * @param args
         */
        public static void main(String[] args) {

            System.out.println("Start:");
            Test_Area ta = new Test_Area();
            ta.getAllAreas();
            ta.deleteAllAreas();
            ta.deleteArea(new AreaName("Your Area"));
            Polygon poly = new Polygon();
            Point[] vertices = new Point[4];
            vertices[0] = new Point();
            vertices[0].setX( 272377.04D);
            vertices[0].setY(6040561.03D);
            vertices[1] = new Point();
            vertices[1].setX( 272256.71D);
            vertices[1].setY(6040625.16D);
            vertices[2] = new Point();
            vertices[2].setX( 272388.94D);
            vertices[2].setY(6040817.55D);
            vertices[3] = new Point();
            vertices[3].setX(0272484.80D);
            vertices[3].setY(6040776.56D);
```

```
            poly.setVertices(vertices);
            AreaName aName = new AreaName("Your Area");
            ta.createArea(poly, aName);
        }
    }
```

D. Java example with AlertAPI

The stubs and skeletons are generated by the IDE tool from the WSDL files.

```
import java.net.MalformedURLException;
import java.rmi.RemoteException;
import java.util.Calendar;
import javax.xml.rpc.ServiceException;
import cellvision.alertix.api.AlertApi;
import cellvision.alertix.api.AlertApiLocator;
import cellvision.alertix.api.AlertApiPortType;
import cellvision.alertix.api.vo.AlertResponse;
import cellvision.alertix.common.domain.ExecuteMode;
import cellvision.alertix.common.domain.api.AlertMsg;
import cellvision.alertix.common.domain.api.AreaName;
import cellvision.alertix.common.domain.api.JobId;
import cellvision.alertix.common.domain.api.TextMessage;
import cellvision.alertix.common.domain.api.WhiteListName;
import cellvision.alertix.common.domain.api.WhiteLists;
import cellvision.alertix.common.vo.Response;

public class Test_Alert {

      private static final String AlertLocation =
"https://access.mobil.telenor.no/alertix/AlertApi";
      private static final String userName = "demo";
      private static final String passWord = "demo";

      private void alert(String message) {

            System.out.println("prepareAlert()");
            AlertApi sa = new AlertApiLocator();
            try {
                  AlertApiPortType aapt = sa.getAlertApiHttpPort(new
java.net.URL(AlertLocation));
                  org.apache.axis.client.Stub stub =
(org.apache.axis.client.Stub)aapt;
                  stub.setUsername(userName);
                  stub.setPassword(passWord);
                  AreaName aName = new AreaName("yourArea");
                  AlertMsg aMessage = new AlertMsg();
                  TextMessage tm = new TextMessage();
                  tm.setText(message);
                  tm.setOa("Your alert message");
                  aMessage.setDefaultMessage(tm);
                  Calendar now = Calendar.getInstance();
                  now.add(Calendar.MINUTE, 10);
                  aMessage.setExpiryTime(now);
                  WhiteListName wln = new WhiteListName();
                  wln.setValue("test_list");
                  WhiteLists wList = new WhiteLists();
                  WhiteListName[] whiteLists = new WhiteListName[1];
                  whiteLists[0] = wln;
                  wList.setWhiteLists(whiteLists);
                  AlertResponse ar = aapt.prepareAreaAlert(aName, aMessage,
wList, null);
                  Integer gc = ar.getCode();
```

```
                    String gm = ar.getMessage();
                    boolean gs = ar.getSuccessful();
                    JobId ji = ar.getJobId();
                    String answer = null;
                    if(ji != null)answer = ji.getValue();
                    // Success
                    System.out.println("Prepare Result: Code:" + gc + " Message:"
+ gm + " Successful:" + gs + " JobId:" + answer);
                    Thread.sleep(5000);
                    Response res = aapt.executePreparedAlert(ji,
ExecuteMode.LIVE);
                    gc = res.getCode();
                    gm = res.getMessage();
                    gs = res.getSuccessful();
                    // Success
                    System.out.println("Execute Result: Code:" + gc + " Message:"
+ gm + " Successful:" + gs );
              } catch (ServiceException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              } catch (RemoteException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              } catch (MalformedURLException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
              }
        }
        /**
         * @param args
         */
        public static void main(String[] args) {

              System.out.println("Start:");
              Test_Alert ta = new Test_Alert();
              ta.alert("Demo Text");
        }
}
```