MANUAL DE UTILIZACIÓN DE LA LIBRERÍA JAR DEL SEM

INFORMACIÓN GENERAL

Proyecto	Dirección técnica - Área Desarrollo - SEM
Referencia	DTEC-ADES-SEM-00003
Fecha	06/10/2010
Versión	1.2.0
Autor	Xilon Solutions
Resumen	Este documento pretende mostrar de forma simple la utilización de las funciones de la librería JAR para acceder al Servicio de Programación de Mensajes del Sistema de Envío Masivo (SEM).

CONTROL DE VERSIONES

VERSIÓN	FECHA	DESCRIPCIÓN DEL CAMBIO
1.1.0	14/06/2010	Versión inicial del manual
1.2.0	06/10/2010	Versión que modifica el saldo de las licencias a decimal al permitir envíos multi-internacionales.

CONTROL DE FIRMAS

PREPARACIÓN	REVISIÓN	APROBACIÓN
Autor: Fecha:	Autor: Fecha:	Autor: Fecha:

TABLA DE CONTENIDO

1. OPERACIONES

Esta librería permite llevar a cabo las siguientes operaciones: Sincronización, Consulta de saldo, Envío, Consulta de licencias, Consulta por petición, Consulta por mensajes, Cancelación de petición y Cancelación por mensajes.

1.1. Inicialización

Para realizar las operaciones se dispone de los métodos de la clase principal de la API del SEM, la SemApi. Para empezar a trabajar con el API del SEM, lo primero que hay que hacer es crear un objeto de la clase SemApi , existen cinco posibles constructores:

1- Constructor simple

SemApi(String license, String user, String pass, String semHost)

2- Constructor con puerto

SemApi(String license, String user, String pass, String semHost, int semPort)

3- Constructor con proxy

SemApi (String license, String user, String pass, String semHost, int semPort, String proxyServer, int proxyPort)

4- Constructor con proxy y autentificación

SemApi (String license, String user, String pass, String semHost, int semPort, String proxyServer, int proxyPort, String proxyUser, String proxyPass)

5- Constructor con puerto y zona horaria

SemApi(String license, String user, String pass, String timezone, String semHost, int semPort)

Definición de los parámetros:

license[String]: Número de la licencia.

user [String]: Nombre de usuario de la licencia.

pass [String]: Clave de la licencia.

semHost [String]: Servidor al que se realiza la petición.

semPort [int]: El puerto al que se realiza la petición.

proxyServer [String]: IP ó nombre del servidor proxy.

proxyPort [int]: Puerto del servidor proxy.

proxyUser [String]: Nombre de usuario del proxy.

proxyPass [String]: Contraseña del proxy.

timezone [String]: Zona horaria en que se desea operar con el servidor. Si no establece, se interpreta como la zona horaria GMT0. Las zonas horarias permitidas se especifican en el apéndice en el apartado 3.4. Zonas horarias permitidas.

A partir del objeto de tipo SemApi se podrá llamar a cada uno de los métodos que ejecutan las operaciones de la API. Todos los métodos que representan operaciones de la API del SEM devuelven un bean de tipo *ApiResponseBean*.

1.1.1. Ejemplos de Inicialización

1- Constructor simple

SemApi(“LSTD020801EALOKOQIVC“, “ uuuu1111“,“pppp1111“, “xx.servicios.mensario.com“)

2- Constructor con puerto

SemApi(“LSTD020801EALOKOQIVC“,“uuuu1111“,“pppp1111, “ xx.servicios.mensario.com“, 0)

3- Constructor con proxy

SemApi(“LSTD020801EALOKOQIVC“, “uuuu1111“, “pppp1111, “xx.servicios.mensario.com“, 0, “192.168.1.2“, 2424)

4- Constructor con proxy y autentificación

SemApi(“LSTD020801EALOKOQIVC“, “uuuu1111“, “pppp1111“, “xx.servicios.mensario.com“, 0, “192.168.1.2“, 2424, “user01“, “pass01“)

5- Constructor con puerto y zona horaria

SemApi(“LSTD020801EALOKOQIVC“, “uuuu1111“, “pppp1111“, “Europe/Madrid“, “xx.servicios.mensario.com“, 0)

1.2. Operación de Sincronización

Permite la sincronización horaria entre el cliente y el servidor. El servidor devuelve la hora en la zona horaria especificada ó en GMT0 en su defecto.

Esta operación se lleva a cabo mediante el uso de las siguientes funciones: SetTimezone (opcional), ExecuteSyncronization, GetResult y GetTimestamp.

1.2.1. Formato Sincronización

Pasos a seguir:

1- SetTimezone(timeZone)

Función opcional para realizar la sincronización. Indica la zona horaria (3.4 Zonas horarias permitidas) en la que se desea recibir la hora del servidor.

Nota: Si se ha utilizado el constructor con puerto y zona horaria no es necesario.

2- ExecuteSyncronization()

Función que devuelve un ApiResponseBean, que permite a través de sus métodos obtener los datos necesarios. En los siguientes pasos se indica que métodos usar.

3- GetResult()

Si el resultado de esta función es OK, continuamos con el paso 4.

4- GetTimestamp()

Esta función nos devuelve la fecha y hora del servidor con formato AAAAMMDDhhmmss. Si se ha realizado el paso 1, en la zona horaria indicada, sino en la zona horaria GMT0.

1.2.2. Ejemplos Sincronización

1.2.2.1. Ejemplo de sincronización no indicando zona horaria

1- ExecuteSyncronization() -> apiResponseBean

2- apiResponseBean.getResult() -> “OK“

3- apiResponseBean.getTimestamp() -> 20100515125210

La fecha y hora actuales del servidor son: 15-05-2010 12:52:10 en la zona horaria GMT0

1.2.2.2. Ejemplo de sincronización indicando zona horaria

1- SetTimezone(“Europe/Madrid”)

2- ExecuteSyncronization() -> apiResponseBean

3- apiResponseBean.getResult() -> “OK“

4- apiResponseBean.getTimestamp() ->20100515135210

La fecha y hora actuales del servidor son: 15-05-2010 13:52:10 en la zona horaria “Europe/Madrid”

1.3. Operación de Consulta de Saldo

Permite consultar el saldo en mensajes de una licencia.

1.3.1. Formato Consulta de Saldo

Pasos a seguir:

1- ExecuteBalanceEnquiry()

Función que devuelve un ApiResponseBean, que permite a través de sus métodos obtener los datos necesarios. En los siguientes pasos se indica que métodos usar.

2- GetResult()

Si el resultado de esta función es OK, continuamos con el paso 3.

3- GetQuantity()

Se obtiene el saldo, número de mensajes en decimal de la licencia.

1.3.2. Ejemplo Consulta de Saldo

1- ExecuteBalanceEnquiry() -> apiResponseBean

2- apiResponseBean .getResult() -> “OK“

3- apiResponseBean .getQuantity() -> 512,0

El saldo de la licencia consultada es de 512,0 mensajes.

1.4. Operación de Envío

Permite la programación de un envío de mensajes. En los envíos se permite enviar mensajes personalizados para cada destinatario, mediante el uso de parámetros en el texto de los mensajes. En el apartado 1.4.2. se muestran ejemplos de los distintos envíos que se pueden realizar.

Esta operación está limitada a 20.000 mensajes por envío, si se sobrepasa esta cantidad la API devuelve un error de sintaxis incorrecta.

1.4.1. Formato Envío

Pasos a seguir:

1- Por cada mensaje a enviar se crea un ApiMessageBean con sus datos, y los añadimos a un ArrayList de objetos ApiMessageBean.

a- Para crear un mensaje (ApiMessageBean) se especifica el remitente, el texto que reciben los destinatarios y la fecha de envío (En GMT0 u opcionalmente en la zona horaria especificada en el paso 2).

Si no se especifica remitente en el mensaje, al destinatario le llegará como “PRIVADO” en el idioma del país destinatario.

Cuando se desee personalizar el contenido del mensaje para cada destinatario (plantillas), se utilizarán parámetros, los parámetros son los elementos personalizables del texto del mensaje. Éstos aparecerán en el texto de la siguiente manera: {#nombre del parámetro}.

Ejemplo de mensaje que utiliza un parámetro:

Text: “Sr. {#apellido} le recordamos su cita … “, con un parámetro con nombre: “apellido” y valor: “Perez”.

Este texto de mensaje se enviará al destinatario como: “Sr. Perez le recordamos su cita…”

Cuando se desee hacer un envío inmediato se puede hacer poniendo como fecha de envío [AAAAMMDDhhmmss] un valor de 14 ceros ("00000000000000"), esto equivale a la fecha y hora actuales.

b- Para especificar los destinatarios del mensaje es necesario crear un ApiRecipientBean por cada destinatario y añadirlo a un ArrayList de objetos ApiRecipientBean.

Un destinatario (ApiRecipientBean) consta de un número de teléfono (prefijo internacional y nº de móvil) y si el mensaje a enviar contiene parámetros en el texto, entonces, también de un hashmap con los nombres de los parámetros y sus valores para dicho destinatario.

2- SetTimezone(timeZone)

Función opcional que permite especificar la zona horaria para el envío de los mensajes. En el apartado 3.4 se indican las zonas horarias permitidas. Si el resultado de esta función es OK, se continúa en el paso 3.

Nota: Si se ha utilizado el constructor con puerto y zona horaria no es necesario.

3- ExecuteSending(ArrayList<ApiMessageBean> sendings)

Función que tiene como parámetro de entrada un arrayList con los mensajes a enviar en la petición y devuelve un ApiResponseBean, que permite a través de sus métodos obtener los datos necesarios. En los siguientes pasos se indica que métodos usar.

4- GetResult ()

Si el resultado de esta función es OK, se continúa en el paso 5.

5- GetChargedQuantity ()

Se obtiene el coste del envío en decimal, es decir el número de mensajes que ha costado.

6- GetRequest()

Se obtiene el identificador de la petición de envío que se acaba de realizar (idRequest).

7- GetIds()

Se obtiene un arrayList con los identificadores de los mensajes enviados en la petición.

1.4.2. Ejemplos de Envío

1.4.2.1. Ejemplo de envío de un mensaje

Este ejemplo muestra el envío a un móvil del siguiente mensaje especificando la zona horaria:

- (34) 666555444: Envío en la fecha: 02-06-2010 10:30:00, del mensaje “Buenos días Sr. Boente, el Dtr. Martinez le desea un Feliz Cumpleaños” con remitente “Dentos”.

1- Se crea un ApiMessageBean al que llamamos message con los datos del mensaje y se añade al ArrayList de objetos ApiMessageBean.

a- Remitente: message.setSender(“Dentos”)

Texto: message.setText“Buenos días Sr. Boente, el Dtr. Martinez le desea un Feliz Cumpleaños”)

Fecha: message.setDate (20100206103000)

b- Destinatario: message.setRecipients(recipients)

1- Número de móvil: recipient.setPhone(“666555444“)

2- Prefijo internacional: recipient.setCode(“34“)

recipients.add(recipient)

c- messages.add(message)

2- SetTimezone(“Europe/Madrid”)

3- ExecuteSending(messages) -> apiResponseBean

4- apiResponseBean .getResult() -> “OK“

5- apiResponseBean .getChargedQuantity() -> 1,0

6- apiResponseBean .getRequest() -> 10250090

Se obtiene el identificador de la petición de envío, **idRequest** = 10250090.

7- GetIds() -> 15000810

Se obtiene el identificador del mensaje enviado: **idSMS** = 15000810.

1.4.2.2. Ejemplo de envío de varios mensajes

Este ejemplo muestra el envío a dos móviles de los siguientes mensajes:

- (34) 666555333: Envío inmediato sin remitente del mensaje “TALLERES PEPE. Buenos días, su vehículo ya está arreglado, puede pasar a recogerlo cuando desee.” (El destinatario recibirá como remitente “PRIVADO”).
- (34) 666111222: Envío inmediato sin remitente del mensaje “TALLERES PEPE. Buenos días, su vehículo ya está arreglado, puede pasar a recogerlo cuando desee.” (El destinatario recibirá como remitente “PRIVADO”).

1- Se crea un ApiMessageBean al que llamamos message con los datos del mensaje.

a- Texto: message.setText(“TALLERES PEPE. Buenos días, su vehículo ya está arreglado, puede pasar a recogerlo cuando desee”)

Fecha: message.setDate (00000000000000)

b- Destinatario: message.setRecipients(recipients)

1- Número de móvil: recipient1.setPhone(“666555333“)

2- Prefijo internacional: recipient1.setCode(“34“)

recipients.add(recipient1)

3- Número de móvil: recipient2.setPhone("666111222")

4- Prefijo internacional: recipient2.setCode("34")

recipients.add(recipient2)

c- messages.add(message)

2- ExecuteSending(messages) -> apiResponseBean

3- apiResponseBean .getResult() -> "OK"

4- apiResponseBean .getChargedQuantity() -> 2,0

5- apiResponseBean .getRequest() -> 10250095

Se obtiene el identificador de la petición de envío, **idRequest** = 10250095.

6- GetIds() -> 15000820 y 15000821

Se obtienen los identificadores de cada mensaje enviado: **idSMS** = 15000820 para el primero e **idSMS** = 15000821 para el segundo.

1.4.2.3. Ejemplo de envío de mensajes con parámetros

Este ejemplo muestra un envío desde la franja horaria "Europe/Madrid" (por ejemplo España), pero sin especificar la zona horaria, es decir las fechas se convierten previamente a GMT0. Envío a cuatro teléfonos móviles de los siguientes mensajes:

- (34) 666111111: Envío inmediato del mensaje "Buenas tardes." Con remitente "Xilon".
- (34) 666111112: Envío inmediato del mensaje "Buenas tardes." Con remitente "Xilon".
- (34) 666111113: Envío planificado para el día 28/06/2010 a las 10h. del mensaje "Hola María feliz 23 cumpleaños". Con remitente "Alonso".
- (34) 666111114: Envío planificado para el día 28/06/2010 a las 10h. del mensaje "Hola Juan feliz 18 cumpleaños". Con remitente "Alonso".

1- En este caso se crearán dos ApiMessageBeans, por haber dos textos distintos: message1 y message2

1.1- Se crea message1 con los datos del mensaje.

a- Remitente: message1.setSender("Xilon")

Texto: message1.setText(“Buenas tardes“)

Fecha: message1.setDate (00000000000000)

b- Destinatario: message1.setRecipients(recipients1)

1- Número de móvil: recipient1.setPhone(“666111111“)

2- Prefijo internacional: recipient1.setCode(“34“)

recipients1.add(recipient1)

3- Número de móvil: recipient2.setPhone(“666111112“)

4- Prefijo internacional: recipient2.setCode(“34“)

recipients1.add(recipient2)

5- messages.add(message1)

1.2- Se crea message2 con los datos del mensaje.

a- Remitente: message2.setSender(“Alonso“)

Texto: message2.setText(“Hola {#nombre} feliz {#años} cumpleaños“)

Fecha: message2.setDate (20100628090000)

b- Destinatario: message2.setRecipients(recipients2)

1- Número de móvil: recipient3.setPhone(“666111113“)

2- Prefijo internacional: recipient3.setCode(“34“)

3- Parámetros (HashMap [nombre_parámetro, valor_parámetro])

parametersHashMap.put(“nombre“, “María“)

parametersHashMap.put(“años“, “23“)

recipient3. setParameters(parametersHashMap)

recipients2.add(recipient3)

4- Número de móvil: recipient4.setPhone(“666111114“)

5- Prefijo internacional: recipient4.setCode(“34“)

6- Parámetros (HashMap [nombre_parámetro, valor_parámetro])

parametersHashMap.put(“nombre“,“ Juan“)

parametersHashMap.put(“años“, “18“)

recipient4. setParameters(parametersHashMap)

recipients2.add(recipient4)

7- messages.add(message2)

2- ExecuteSending(messages) -> apiResponseBean

3- apiResponseBean .getResult() -> “OK“

4- apiResponseBean .getChargedQuantity() -> 4,0

5- apiResponseBean .getRequest() -> 10250100

Se obtiene el identificador de la petición de envío, **idRequest** = 10250100.

6- GetIds() -> 15000901, 15000902, 15000903 y 15000904

Se obtienen los identificadores de cada mensaje enviado: **idSMS** = 15000901 para el mensaje al móvil: 666111111, **idSMS** = 15000902 para el mensaje al móvil: 666111112, **idSMS** = 15000903 para el mensaje al móvil: 666111113 e **idSMS** = 15000904 para el mensaje al móvil: 666111114.

1.5. Operación de Consulta de Licencias

Permite averiguar el saldo en mensajes, el tipo (Estándar/Demostración) y el estado (válida, incorrecta, deshabilitada ó no existe) de las licencias consultadas.

Esta operación está limitada a 1.000 licencias, si se sobrepasa esta cantidad la API devuelve un error de sintaxis incorrecta.

1.5.1. Formato Consulta de Licencias

Pasos a seguir:

1- Por cada licencia que se desee enviar se crea un ApiLicenseBean con sus datos, y los añadimos a un ArrayList de objetos ApiLicenseBean.

Para realizar la consulta se debe autentificar cada licencia, para ello es necesario indicar el número de licencia, el usuario y la contraseña.

2- ExecuteLicensesQuery(ArrayList< ApiLicenseBean> licenses)

Función que tiene como parámetro de entrada un arrayList con las licencias a consultar y devuelve un ApiResponseBean, que permite a través de sus métodos obtener los datos necesarios. En los siguientes pasos se indica que métodos usar.

3- GetResult()

Si el resultado de esta función es OK, continuamos con el paso 4.

4- GetLicenseData()

Función que devuelve un arrayList de ApiLicenseDataBean, que permite a través de sus métodos obtener los datos para cada licencia. En los siguientes pasos se indica que métodos usar.

5- GetNumber()

Este método de ApiLicenseDataBean devuelve el número de la licencia consultada.

6- GetQuantity()

Este método de ApiLicenseDataBean devuelve el saldo en mensajes de la licencia consultada.

7- GetType()

Este método de ApiLicenseDataBean devuelve el tipo de la licencia consultada.

8- GetStatus()

Este método de ApiLicenseDataBean devuelve el estado de la licencia consultada.

1.5.2. Ejemplos de Consulta de Licencias

Consulta de una licencia con Nº de serie: LSTD020801EKLOKOQIVB, usuario: ai9gq561 y clave: ai9gq123.

1- Se crea un ApiLicenseBean al que llamamos license con sus datos, y los añadimos a un ArrayList de objetos ApiLicenseBean.

a- Número de licencia: license.setNumber(“LSTD020801EKLOKOQIVB”)

b- Usuario de licencia: license.setUser(“ai9gq561”)

c- Contraseña de licencia: license.setPass(“ai9gq123”)

Licenses.add(license)

2- ExecuteLicensesQuery(ArrayList< ApiLicenseBean> licenses) -> apiResponseBean

3- apiResponseBean.getResult() -> “OK“

4- apiResponseBean.getLicenseData() -> apiLicenseDataBean

5- apiLicenseDataBean.getNumber () -> “LSTD020801EKLOKOQIVB”

Se obtiene el número de la licencia, “LSTD020801EKLOKOQIVB”.

6- apiLicenseDataBean.getQuantity () -> 25,0

El saldo de la licencia es de 25,0 mensajes.

7- apiLicenseDataBean.getType () -> “STD“

La licencia es de tipo estándar.

8- apiLicenseDataBean.getStatus () -> “OK“

La licencia es válida.

Consulta de varias licencias: Licencia 1 con Nº de serie: LSTD020801AAAAAAAAAA, usuario: aaaa1111 y clave: aaaa2222 y Licencia 2 con Nº de serie: LSTD021001BBBBBBBBBB, usuario: bbbb1111 y clave: bbbb2222

1- Se crea un ApiLicenseBean por cada licencia que se desee consultar con sus datos, y los añadimos a un ArrayList de objetos ApiLicenseBean.

1.1- Se crea license1 con los datos de la primera licencia

a- Número de licencia: license1.setNumber(“LSTD020801AAAAAAAAAA”)

b- Usuario de licencia: license1.setUser(“aaaa1111”)

c- Contraseña de licencia: license1.setPass(“aaaa2222”)

licenses.add(license1)

1.2- Se crea license2 con los datos de la segunda licencia

a- Número de licencia: license2.setNumber(“LSTD021001BBBBBBBBBB”)

b- Usuario de licencia: license2.setUser(“bbbb1111”)

c- Contraseña de licencia: license2.setPass(“bbbb2222”)

licenses.add(license2)

2- ExecuteLicensesQuery(ArrayList< ApiLicenseBean> licenses) -> apiResponseBean

3- apiResponseBean.getResult() -> “OK“

4- apiResponseBean.getLicenseData() -> licenseData1 y licenseData2

Se obtiene un arrayList de ApiLicenseDataBeans con un ApiLicenseDataBean por cada una de las licencias.

5- licenseData1.getNumber () -> “LSTD020801AAAAAAAAAA”

Se obtiene el número de la licencia, “LSTD020801AAAAAAAAAA”

6- licenseData1.getQuantity () -> 2,0

El saldo de la licencia LSTD020801AAAAAAAAAA, es de 2 ,0 mensajes.

7- licenseData1.getType () -> “DEM“

El tipo de la licencia LSTD020801AAAAAAAAAA es demostración.

8- licenseData1.getStatus () -> “KO-DIS“

La licencia LSTD020801AAAAAAAAAA está caducada ó bloqueada

9- licenseData2.getNumber () -> “LSTD021001BBBBBBBBBB”

Se obtiene el número de la licencia, “LSTD021001BBBBBBBBBB”

10- licenseData2.getQuantity () -> 4.510,0

El saldo de la licencia: LSTD021001BBBBBBBBBB, es de 4.510,0 mensajes.

11- licenseData2.getType () -> “STD“

El tipo de la licencia LSTD021001BBBBBBBBBB, es estándar.

12- licenseData2.getStatus () -> “OK“

La licencia LSTD021001BBBBBBBBBB, es válida.

1.6. Operación de Consulta por Petición de Envío.

Permite averiguar el estado de los mensajes de una petición de envío dada.

1.6.1. Formato Consulta por Petición de Envío

Pasos a seguir:

1- ExecuteStatusQueryByIdRequest(idRequest)

Función que recibe como parámetro el identificador de la petición de envío (idRequest) que se desea consultar y devuelve un ApiResponseBean, que permite a través de sus métodos obtener los datos necesarios. En los siguientes pasos se indica que métodos usar.

2- GetResult ()

Si el resultado de esta función es OK, continuamos con el paso 3.

3- GetMsgStatus()

Se obtiene un HashMap con el estado de los mensajes como clave y un ArrayList con los identicadores de los mensajes como valor.

1.6.2. Ejemplo Consulta por Petición de Envío

Consulta por petición al envío realizado en el segundo ejemplo de la operación envío, cuyo idRequest es 10250095.

1- ExecuteStatusQueryByIdRequest(10250095) -> apiResponseBean

2- apiResponseBean.getResult () -> “OK“

3- apiResponseBean.getMsgStatus () -> msgStatusHashMap

El hashMap contiene un único estado: msgStatusHashMap.get(“CMS-OK”) = 15000820 y 15000821

Se obtiene que, en estado “CMS-OK “(mensaje enviado correctamente) están los mensajes con **idSMSs =** 15000820 y 15000821.

1.7. Operación de Consulta de Envío por mensajes.

Permite averiguar el estado de los mensajes consultados.

Esta operación está limitada a 20.000 mensajes, si se sobrepasa esta cantidad la API devuelve un error de sintaxis incorrecta.

1.7.1. Formato Consulta de Envío por mensajes

Pasos a seguir:

1- Se crea un ArrayList de enteros con los identificadores de los mensajes a consultar.

2- ExecuteStatusQueryByIdMessages (ArrayList<Integer> idMessages)

3- GetResult()

Si el resultado de esta función es OK, continuamos hasta el paso 4.

Si el resultado es KO-UNK-MSGS (Error identificadores de mensajes desconocidos), vamos al paso 5.

4- GetMsgStatus()

Se obtiene un HashMap con el estado de los mensajes como clave y un ArrayList con los identificadores de los mensajes como valor.

5- GetIds()

Se obtiene un ArrayList con los identificadores de los mensajes inválidos.

1.7.2. Ejemplos de Consulta de Envío por mensajes

Consulta de envío por mensajes, en este ejemplo se van a consultar los mensajes enviados en los ejemplos con identificadores: 15000901 y 15000904.

1- Se crea un ArrayList de enteros (idMessages) con los identificadores de los mensajes a consultar.

idMessages.add(15000901)

idMessages.add(15000904)

2- ExecuteStatusQueryByIdMessages (idMessages)

3- GetResult() -> “OK“

4- GetMsgStatus() -> msgStatusHashMap

El hashMap contiene dos estados: msgStatusHashMap.get(“CMS-OK”) = 15000901 y msgStatusHashMap.get(“CMS- PEND”) = 15000904.

Se obtiene que, en estado CMS-OK (mensaje enviado correctamente) está el mensaje con **idSMS =** 15000901, y que en estado CMS-PEND (mensaje pendiente de enviar) está el mensaje con **idSMS =** 15000904.

Consulta de envío por mensajes, en este ejemplo se van a consultar los mensajes con identificadores: 15000820 y 13000000. (El segundo es inválido)

1- Se crea un ArrayList de enteros (idMessages) con los identificadores de los mensajes a consultar.

idMessages.add(15000820)

idMessages.add(13000000)

2- ExecuteStatusQueryByIdMessages (idMessages)

3- GetResult() -> “KO-UNK-MSGS“

Devuelve KO-UNK-MSGS: Error identificadores de mensajes desconocidos

4- GetIds() -> 13000000

Se obtiene un identificador de mensaje inválido, **idSMS** = 13000000.

1.8. Operación de Cancelación por Petición de Envío.

Permite cancelar todos los mensajes que se encuentren en el estado PENDIENTE de una petición dada, devolviendo al usuario el saldo de mensajes correspondiente en la licencia con la que se realizó el envío.

Para optimizar el resultado de la operación, solo se devolverán los identificadores de los mensajes cancelados cuando se produzca una cancelación parcial de la petición, es decir, si la petición se cancela totalmente ó no se pueda cancelar ningún mensaje, estos no se obtienen.

1.8.1. Formato Cancelación por Petición de envío

Pasos a seguir:

1- ExecuteCancellationByIdRequest (idRequest)

Función que recibe como parámetro el identificador de la petición que se desea cancelar y devuelve un ApiResponseBean.

2- GetResult()

Si el resultado de esta función es OK, continuamos en el paso 3

3- GetRefundedQuantity()

Se obtiene el saldo en mensajes que se ha devuelto por la cancelación.

4- GetTotal()

Se obtiene el total de mensajes que contiene la petición de envío a cancelar.

5- GetCancel()

Se obtienen la cantidad de mensajes que se han podido cancelar de la petición.

6- GetIds()

Si la cancelación de la petición ha sido parcial, se obtiene un ArrayList de enteros con los identificadores de los mensajes que se han cancelado.

1.8.2. Ejemplos de Cancelación por Petición de envío

Cancelación por petición, en este ejemplo se van a cancelar los mensajes de la petición realizada en los ejemplos con identificador: 10250090

1- ExecuteCancellationByIdRequest (10250090) -> apiResponseBean

2- apiResponseBean.getResult() -> "OK"

3- apiResponseBean.getRefundedQuantity() -> 1,0

4- apiResponseBean.getTotal() -> 1

La petición a cancelar está compuesta de un único mensaje.

5- apiResponseBean.getCancel() -> 1

Se ha podido cancelar un mensaje. La petición ha sido cancelada en su totalidad.

Cancelación por petición, en este ejemplo se van a cancelar los mensajes de la petición realizada en los ejemplos con identificador: 10250100

1- ExecuteCancellationByIdRequest (10250100) -> apiResponseBean

2- apiResponseBean.getResult() -> "OK"

3- apiResponseBean.getRefundedQuantity() -> 2,0

Se obtiene el sado devuelto por la cancelación: 2,0.

4- apiResponseBean.getTotal() -> 4

La petición a cancelar está compuesta de cuatro mensajes.

5- apiResponseBean.getCancel() -> 2

Se han podido cancelar dos mensajes.

6- apiResponseBean.getIds() -> 15000901 y 15000902

Devuelve los identificadores de los mensajes cancelados. Se han cancelado dos el primero con **idSMS** = 15000901 y el segundo con **idSMS** = 15000902.

1.9. Operación de Cancelación de Mensajes

Permite cancelar los mensajes detallados en la operación que se encuentren en estado PENDIENTE, devolviendo al usuario la cantidad de mensajes que se han cancelado.

Esta operación está limitada a 20.000 mensajes, si se sobrepasa esta cantidad la API devuelve un error de sintaxis incorrecta.

Para optimizar el resultado de la operación, solo se devolverán los identificadores de los mensajes cancelados cuando se produzca una cancelación parcial, es decir, si se cancelan todos los mensajes ó ninguno los identificadores no se obtienen.

1.9.1. Formato Cancelación de Mensajes

Pasos a seguir:

1- Se crea un ArrayList de enteros con los identificadores de los mensajes que se desea cancelar.

2- ExecuteCancellationByIdMessages (ArrayList<Integer> idsMessages)

Función que recibe como parámetro de entrada un arrayList de enteros con los identificadores de los mensajes a cancelar y devuelve un ApiResponseBean.

3- GetResult()

Si el resultado de esta función es OK, continuamos con el paso 4.

Si el resultado es KO-UNK-MSGS (Error identificadores de mensajes desconocidos), vamos al paso 6.

4- GetRefundedQuantity()

Se obtiene el saldo en mensajes que se ha devuelto por la cancelación.

5- GetTotal()

Se obtiene el total de mensajes que contiene la petición de envío a cancelar.

6- GetCancel()

Se obtiene la cantidad de mensajes que se han podido cancelar de la petición.

7- GetIds()

Si la cancelación de la petición ha sido parcial, se obtiene un ArrayList de enteros con los identificadores de los mensajes que fueron cancelados. Si la operación ha fallado por contener mensajes inválidos, con este método se obtienen los identificadores de estos mensajes.

1.9.2. Ejemplos de Cancelación de Mensajes

Cancelación de mensajes, en este ejemplo se van a cancelar los mensajes de los ejemplos con identificadores: 15000821 y 15000902.

1- Se crea un ArrayList de enteros (idsMessages) con los identificadores de los mensajes que se desea cancelar.

idsMessages.add(15000821)

idsMessages.add(15000902)

2- ExecuteCancellationByIdMessages (idsMessages) -> apiResponseBean

3- apiResponseBean.getResult () -> “OK“

4- apiResponseBean.getRefundedQuantity() -> 2,0

Se obtiene el saldo en mensajes que se ha devuelto en la cancelación: 2,0

5- apiResponseBean.getTotal() -> 2

Se obtiene el total de mensajes a cancelar, 2.

6- apiResponseBean.getCancel() -> 2

Se obtienen la cantidad de mensajes que se han podido cancelar, 2. Se han podido cancelar todos los mensajes.

Cancelación de mensajes, en este ejemplo se van a cancelar los mensajes de los ejemplos con identificadores: 15000821 y 15.

1- Se crea un ArrayList de enteros (idsMessages) con los identificadores de los mensajes que se desea cancelar.

idsMessages.add(15000821)

idsMessages.add(15)

2- ExecuteCancellationByIdMessages (idsMessages) -> apiResponseBean

3- apiResponseBean.getResult () -> “KO-UNK-MSGS“

Devuelve KO- UNK-MSGS, error por identificadores de mensajes desconocidos.

7- apiResponseBean.getIds() -> 15

Devuelve los identificadores de los mensajes inválidos, **idSMS** = 15.

2. BEANS

2.1. ApiResponseBean

Definición de los campos de apiResponseBean:

result [String]: Código de respuesta de la operación. Códigos posibles:

OK : Bien.

KO-INV-AUTH : Error autentificación inválida.

KO-BLK-LIC : Error licencia bloqueada.

KO-EXP-LIC : Error licencia expirada.

KO-INV-PHON : Error teléfono inválido.

KO-INV-CODE : Error prefijo inválido.

KO-UNK-REQU : Error petición desconocida.

KO-UNK-MSGS : Error mensajes desconocidos.

KO-EXC-AMNT : Error saldo insuficiente.

KO-INV-TIME : Error mensaje fuera de rango temporal.

KO-INV-SINT : Error sintaxis incorrecta.

KO-INV-TZ : Error zona horaria inválida.

KO-DEN-CONN : Error servidor no conectable.

KO-DIS-SERV : Error servicio no disponible temporalmente.

KO-INT-ERR : Error interno.

Debido a que el funcionamiento del protocolo está embebido en la propia librería JAR que accede a la API del SEM, existen otros dos códigos de respuesta adicionales: KO-INV-TASK (Error por operación inválida) y KO-INV-XML (Error por xml inválido) que no pueden salir nunca.

timestamp [Date]: Fecha y hora del servidor en formato AAAAMMDDhhmmss, en la zona horaria especificada ó en su defecto GMT0.

quantity [Decimal]: Saldo, número de mensajes de la licencia.

chargedQuantity [Decimal]: Saldo, número de mensajes de la licencia que ha costado el envío.

refundedQuantity [Decimal]: Saldo, número de mensajes de la licencia que se ha devuelto al realizar una cancelación.

request [Integer]: Identificador de la petición del envío.

total [Integer]: Número de mensajes de los que consta la cancelación.

cancel [Integer]: Número de mensajes cancelados.

ids [ArrayList<Integer>]: ArrayList con los identificadores de los mensajes.

msgStatus [HashMap<String, ArrayList<Integer>>]: HashMap en el que la clave es en código de estado de mensaje y el valor un ArrayList con los identificadores de mensaje que tienen ese estado. Códigos de estado posibles:

CMS-PEND : El mensaje está pendiente de ser enviado.

CMS-CANC : El mensaje ha sido cancelado.

CMS-SEND : El mensaje ha sido enviado, está pendiente de confirmación.

CMS-OK : El mensaje ha llegado al destinatario.

CMS-KO-INV : El envío del mensaje ha fallado. El teléfono del destinatario es inválido.

CMS-KO-EXP : El envío del mensaje ha fallado. Se ha excedido el tiempo de envío, teléfono apagado, sin cobertura.

CMS-KO-RES : El envío del mensaje ha fallado. El acceso al destinatario está restringido.

CMS-KO-UNK : Sin información. No se ha recibido confirmación.

licenseData [ArrayList<ApiLicenseDataBean>]: ArrayList de datos de licencias (número, saldo, tipo y estado).

2.1.1. Métodos de ApiResponseBean

getResult(): Obtiene el resultado de la operación.

getTimestime(): Obtiene la fecha y hora del SEM.

getQuantity(): Obtiene el saldo, cantidad de mensajes de una licencia en decimal.

getChargedQuantity(): Obtiene el saldo, cantidad de mensajes que ha costado el envío.

getRefundedQuantity(): Obtiene el saldo, cantidad de mensajes que se ha devuelto en una cancelación.

getRequest(): Obtiene el identificador de petición del SEM.

getTotal(): Obtiene el total de mensajes a cancelar.

getCancel(): Obtiene el número de mensajes que se han podido cancelar.

getIds(): Obtiene los identificadores de los mensajes.

getMsgStatus(): Obtiene un HashMap en el que la clave es el estado y el valor un ArrayList con los identificadores de mensaje que tienen ese estado, es decir, los mensajes consultados agrupados por estado.

getLicenseData(): Obtiene un ArrayList de ApiLicenseDataBean.

2.2. ApiMessageBean

recipients [ArrayList<ApiRecipientBean>]: ArrayList con los datos de los destinatarios de los mensajes (teléfono, prefijo y parámetros).

sender [String GSM (0-11)]: Remitente del mensaje. Si el elemento sender no se recibe ó no contiene ningún texto, al destinatario le llegará como PRIVADO en el idioma del país destinatario. Los caracteres permitidos en el remitente se especifican en el apéndice en el apartado 9.1. Caracteres válidos para el remitente.

text [String GSM (1-160)]: Texto del mensaje. Cuando se desee enviar un mensaje con parámetros, estos aparecerán en el texto de la siguiente manera: {#nombre del parámetro}. Los caracteres permitidos en el texto se especifican en el apéndice en el apartado 9.2. Caracteres válidos para el texto.

Ejemplo: El mensaje siguiente utiliza el parámetro apellido en el texto: Sr. {#apellido} le recordamos su cita …

date [String]: Fecha y hora en el formato AAAAMMDDhhmmss, a la que debe enviarse el mensaje. Un valor de 14 ceros ("00000000000000") equivale a la fecha y hora actuales.

2.2.1. Métodos de ApiMessageBean

setRecipients(ArrayList<ApiRecipientBean>): Establece los destinatarios de un mensaje.

setSender(String): Establece el remitente de un mensaje.

setText (String): Establece el texto de un mensaje.

setDate(String): Establece la fecha y hora de envío de un mensaje.

2.3. ApiRecipientBean

phone [String]: El número del móvil destinatario.

cod [String]: Prefijo telefónico del país del destinatario. Ejemplo: Para un mensaje enviado a un móvil español es 34.

Parameters [HashMap<String, String>]: Parámetros a sustituir en el texto del mensaje para este destinatario. Donde la clave del hashmap indica el nombre del parámetro en el texto del mensaje y el valor indica por qué se va a sustituir.

2.3.1. Métodos de ApiRecipientBean

setPhone(String): Establece el número de teléfono móvil.

setCode(String): Establece el código del país.

setParameters (HashMap<String, String>): Establece los parámetros y los valores por los que van a ser sustituidos en el texto.

2.4. ApiLicenseBean

number [String]: Número de la licencia.

user [String]: Nombre de usuario de la licencia

pass [String]: Contraseña de la licencia

2.4.1. Métodos de ApiLicenseBean

getNumber(): Obtiene el número de licencia.

setNumber(String): Establece el número de licencia.

getUser(): Obtiene el nombre de usuario de la licencia.

setUser(String): Establece nombre de usuario de la licencia.

getPass(): Obtiene la contraseña de la licencia.

setPass(String): Establece la contraseña de la licencia.

2.5. ApiLicenseDataBean

number [String]: Número de la licencia.

status [String]: Código del estado de la licencia. Existen los siguientes:

OK: La licencia es válida.

KO-INV : La licencia consultada no existe.

KO-AUTH : La autentificación de la licencia es incorrecta. El usuario y/ó la clave de la licencia no son correctos.

KO-DIS : La licencia está deshabilitada.

quantity [Decimal]: Saldo, número de mensajes de la licencia en el momento de la consulta.

type [String]: Tipo de la licencia, puede ser estándar ó demostración. Los códigos a obtener son los siguientes:

STD: Tipo estándar.

DEM: Tipo demostración.

2.5.1. Métodos de ApiLicenseDataBean

getNumber(): Obtiene el número de licencia.

getStatus(): Obtiene el estado de licencia.

getQuantity(): Obtiene el saldo de licencia si el estado de la licencia es OK ó KO-DIS.

getType(): Obtiene el tipo de la licencia si el estado es OK, KO-AUTH y KO-DIS.

3. APENDICE

El SEM utiliza en la confección de los SMS el alfabeto GSM, es decir, los mensajes solo admiten caracteres existentes en este alfabeto, todos los demás serán considerados caracteres inválidos. Si un mensaje contiene algún carácter inválido no se podrá enviar, se considera inválido.

Una petición de envío con algún mensaje inválido provoca que como respuesta del SEM se obtenga un error de sintaxis incorrecta (KO-INV-SINT). Esto mismo ocurre, si la petición contiene algún mensaje cuyo remitente ó texto exceda del tamaño máximo de caracteres permitido, que es 11 y 160 respectivamente.

3.1. Caracteres válidos para el remitente

El remitente del mensaje a enviar solo admite los siguientes caracteres GSM:

Letras mayúsculas y minúsculas del alfabeto inglés:

ABCDEFGHIJKLMNOPQRSTUVWYXZabcdefghijklmnopqrstuvwxyz

El carácter espacio.

Números del 0 al 9:

0123456789

3.2. Caracteres válidos para el texto

El texto del mensaje además de los caracteres del alfabeto estándar GSM también admite caracteres del alfabeto avanzado. Estos últimos necesitan un tratamiento especial a la hora de incluirlos en el contenido del texto ya que cuentan como dos caracteres a la hora de realizar el envío (el tamaño máximo de permitido en el texto es de 160 caracteres).

3.3. Caracteres del alfabeto GSM

Los caracteres permitidos del alfabeto GSM son los siguientes, se incluyen los del alfabeto avanzado GSM (marcados con otro color y un asterisco). Estos caracteres ocupan dos bytes en lugar de uno:

Caracteres Alfanuméricos																												
Números del 0 al 9:	0	1	2	3	4	5	6	7	8	9																		
Letras Mayúsculas del alfabeto inglés:	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z		
Letras Minúsculas del alfabeto inglés:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z		
Letras Europeas																												
Mayúsculas:	Ä	Ö	Ü								É		Ø		Å		Æ		Ñ		Ç							

Minúsculas:	ä	ö	ü		à	è	ì	ò	ù		é		ø		å		æ		ñ				§		ß

Letras Griegas									
Mayúsculas:	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Φ	Ψ	Ω

Signos de Puntuación			
Espacio		Subrayado	_
Punto	.	Dos Puntos	:
Coma	,	Punto y Coma	;
Comilla simple	'	Comilla doble	"
Interrogación	?	Interrogación de apertura	¿
Admiración	!	Admiración de apertura	¡

Signos Matemáticos, de Moneda y Otros			
Suma	+	Resta (guión)	-
Multiplicación (asterisco)	*	División (barra)	/
Igual	=	* Barra invertida	\
Menor que	<	Mayor que	>
Paréntesis de apertura	(	Paréntesis de cierre	)
* Corchete de apertura	[	* Corchete de cierre	]
* Llave de apertura	{	* Llave de cierre	}
* Acento circunflejo	^	* Acento nasal (tilde)	~
* Barra horizontal	|	Arroba	@
Porcentaje	%	Almohadilla	#
Ampersand	&	Símbolo de moneda	¤
Dólar	$	Libra Esterlina	£
* Euro	€	Yen	¥

Caracteres de Control	
Nueva Línea:	`ASCII y GSM: 0x0A, "\n"`
Retorno de Carro:	`ASCII y GSM: 0x0D, "\r"`
* Salto de página:	`ASCII: 0x0C`, en desuso
Carácter de 'escape':	`ASCII y GSM: 0x1B`, permite el uso de las "tablas de extensión"

3.4. Zonas horarias permitidas

El SEM permite la sincronización y el envío de mensajes a países en diferentes zonas horarias. A continuación se muestran las zonas horarias admitidas:

Códigos admitidos como zona horaria	Descripción
GMT0	GMT+00:00
Europe/Madrid	Hora de España (excepto islas Canarias).
Europe/Lisbon	Hora de Portugal (excepto Azores).
Europe/Andorra	Hora de Andorra.
Atlantic/Canary	Hora de las islas Canarias.
Atlantic/Azores	Hora de las islas Azores.
Atlantic/Madeira	Hora de Madeira.