Mobyt

SMS Gateway Java SDK

TABLE DES MATIERES

INTRODUCTION

Les serveurs Mobyt mettent à disposition des développeurs des interfaces de communication HTTP pour l'envoi et la réception de messages SMS permettant la vérification de la validité des numéros de téléphone et la consultation des données concernant l'état des messages et l'historique SMS.

CONFIGURATION DE LA LIBRAIRIE

L'archive contient les éléments suivants :

- sac/ : directory contenant la source java du sdk
- src/sdk.properties : fichier de configuration de la librairie en format java properties; les options de configuration sont: hostname, qui permet de modifier l'host auquel seront envoyées les requêtes; connection_type, qui précise le type de connexion utilisé; username et password de son compte mobyt;
- src/nations.properties : permet de préciser la nation et le préfixe. Le format est iso639_iso3166=préfixe (exemple: fr_FR=+33)
- mobyt-sdk.jar : librairie préremplie

La librairie téléchargeable se trouve à cette adresse :
http://www.mobyt.fr/pdf/sdk/sms_gateway_java.zip

CRÉATION D'UNE CONNEXION

Toutes les opérations (envoi de SMS, requêtes pour obtenir l'état des messages, etc.) se réalisent par l'interface SMSCConnection. Il est possible d'obtenir un objet SMSCConnection par le biais d'une requête:

SMSCConnectionFactory.openConnection()
SMSCConnectionFactory.openConnection(String login, String password) qui permet de préciser un nom d'utilisateur et un mot de passe spécifiques pour la connexion.

Pour fermer une connexion, utilisez le système logout () de l'interface SMSCConnection.
Même s'il n'est pas nécessaire, nous conseillons de créer un objet SMSConnection au démarrage de l'application et de le fermer à la sortie de l'application.

Exemple:

```
SMSCConnection smsc_connection = null;
try {
    smsc_connection = SMSCConnectionFactory.openConnection();
} catch (SMSCException smsc_ex) {
    System.err.println("error connecting with mobyt: "+smsc_ex.getMessage());
    return;
}
// connection operations
// ...
try {
    smsc_connection.logout();
} catch (SMSCRemoteException smscr_ex) {
    System.err.println("error while closing connection with mobyt: "+smscr_ex.getMessage());
}
```

ENVOI DE SMS

La classe SMS contient toutes les données nécessaires pour envoyer un message SMS par le gateway mobyt.
Une fois créé un objet SMS, il est nécessaire de préciser:

- type de SMS: TOP, DIRECT (méthode setSms_type(SMSType));
- destinataires: numéro de téléphone des destinataires du message en format international; pour ajouter un numéro à la liste des destinataires, utiliser la méthode addSmsRecipient(String) ou addSmsRecipient(SMSRecipient);
- message: la méthode setMessage(String) permet de définir le corps du message à envoyer, maximum 160 caractères pour un seul SMS, jusqu'à 1000 caractères pour un SMS long (concaténé). Attention: les caractères ^ { [~] | € sont calculés comme 2

caractères; il est possible d'utiliser la méthode count_smss() pour connaître le nombre de SMS que vous souhaitez envoyer;
- expéditeur: si vous souhaitez personnaliser l'expéditeur du SMS; maximum 16 caractères pour les numéros de téléphone en format international ou 11 caractères pour les suites de caractères alphanumériques (ex: "+333456789012345" ou "NomSociete"), méthode setSmsSender(String) ou setSmsSender(SMSSender).

Vous pouvez également définir en option :
- la date et l'heure d'un envoi programmé (méthode setScheduled_delivery(Date));
- une chaîne alphanumérique à associer à l'envoi, indispensable pour obtenir ensuite les informations concernant l'état des messages (méthode setOrder_id(String)). La longueur maximum est de 32 caractères, tous les caractères excédents ne seront pas considérés.

Si le Order_id n'est pas indiqué, le serveur va en générer un unique.
Le Order_id peut être utilisé dans les 30 jours suivants pour demander le status du message.

Exemple:
```
try {
    SMS sms = new SMS();
    sms.setSms_type(SMSType.TOP);
    sms.addSmsRecipient("+336061234567");
    sms.addSmsRecipient("+336109876543");
    sms.setMessage("hello world!");
    sms.setSms_sender("Expéditeur");
    sms.setImmediate(); // or sms.setScheduled_delivery(java.util.Date)
    sms.setOrder_id("12345");
    SendResult result = connection.sendSMS(sms);
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
} catch (SMSCException smsce) {
    System.out.println("Exception creating message: "+smsce.getMessage());
}
```

ÉTAT DES MESSAGES

Après l'envoi des messages, il est possible d'en demander l'état en utilisant la méthode getMessageStatus(String) de l'objet SMSConnection. En précisant le id du SMS, il est possible de demander au serveur le résultat de l'envoi – une List<MessageStatus> est restituée; tous les objet du type MessageStatus contiennent :

- le numéro de téléphone du destinataire (méthode getRcpt_number());
- l'état de l'envoi (getStatus());
- date et heure de réception du message, si le SMS a été reçu correctement par le destinataire (méthode getDeliveryDate()).

L'enum Status peut prendre les valeurs suivantes:

- SCHEDULED // programmé, pas encore envoyé
- SENT // envoyé, il n'attend pas de delivery
- DLVRD // SMS reçu correctement
- ERROR // erreur dans l'envoi du SMS
- TIMEOUT // après 48h l'opérateur n'a pas fourni d'informations
- TOOM4NUM // trop de SMS pour le même destinataire dans les dernières 24 heures
- TOOM4USER // trop de SMS envoyés par l'utilisateur pendant les dernières 24 heures
- UNKNPFX // préfixe SMS invalide ou inconnu
- UNKNRCPT // numéro de téléphone du destinataire invalide ou inconnu
- WAIT4DLVR // message envoyé en attente de delivery
- WAITING // en attente, pas encore envoyé
- UNKNOWN // état inconnu

Si vous n'avez pas besoin de gérer en détails les éventuels erreurs d'envoi, l'enum Status fournit une méthode boolean isError() qui renvoi true si le message est en erreur.

Exemple:

```
try {
    List<MessageStatus> smsstatus = connection.getMessageStatus("1234");
    for (MessageStatus message_status : smsstatus) {
            System.out.print("destinatario :"+message_status.getRcpt_number());
            if (message_status.getStatus().isError()) {
                    System.out.println("error!");
            } else {
                    if (message_status.getStatus()==MessageStatus.Status.DLVRD) {
                            System.out.println("delivered.");
                    } else {
                            System.out.println("waiting...");
```

```
                }
            }
        }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

ANNULATION DES ENVOIS PROGRAMMÉS

La méthode removeScheduledSend(String) de l'objet SMSCConnection demande au serveur mobyt d'annuler un envoi programmé antérieurement en configurant le paramètre scheduled_delivery au moment de l'envoi du message.

Exemple :

```
try {
    connection.removeScheduledSend("12345");
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

HISTORIQUE DES MESSAGES

La méthode getSMSHistory() de l'objet SMSCConnection demande au serveur mobyt la liste des SMS envoyés dans un laps de temps spécifique; une List<SentSMS> est renvoyée.

Exemple :

```
try {
    List<SentSMS> sent_smss = connection.getSMSHistory(
            new Date(System.currentTimeMillis()-10000000000L), new Date());
    for (SentSMS sent_sms : sent_smss) {
            System.out.println("SMS "+sent_sms.getOrder_id()+", tipo
            SMS"+sent_sms.getSms_type().toString()+" creato il "+sent_sms.getCreate_time() +"
            inviato a "+sent_sms.getRecipients_count()+" destinatari");
    }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

CRÉDIT SMS DISPONIBLE

La méthode getCredits() de l'objet SMSCConnection permet de demander au serveur mobyt la disponibilité de crédit; une List<Credit> est restituée. Tous les objets Credit contiennent: le type de SMS, la nation (null si le crédit est international) et la quantité de crédit disponible.

Exemple :

```
try {
    List<Credit> credits = connection.getCredits();
    for (Credit credit : credits) {
            if (credit.getNation()!=null && credit.getNation().equals("IT")) {
                    if (credit.getCreditType() == CreditType.TOP) {
                            System.out.println("I can send "+credit.getCount()+" SMS with
                            customized sender ID to France.");
                            break;
                    }
            }
    }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

LECTURE DES MESSAGES REÇUS VIA LE SERVICE DE RÉCEPTION SMS

L'utilisateur propriétaire d'un ou plusieurs services de réception, dédiés ou partagés qui décide de demander les messages au serveur mobyt, peut utiliser indifféremment une des trois méthodes mises à disposition de l'objet SMSConnection; chaque méthode restitue une List<SMS_MO>:
getNewSMS_MOs(), qui demande au serveur tous les “nouveaux” messages, c'est-à-dire, tous les messages reçus après la dernière requête (attention: cette méthode doit être activée par Mobyt; si vous souhaitez utiliser cette méthode, contactez-nous à l'adresse info@mobyt.fr). Pour l'intégration dans une application nous conseillons d'utiliser cette méthode.
getSMS_MOHistory(Date,Date) demande au serveur tous les SMS reçus pendant un laps de temps spécifique.
getSMS_MOById(long) demande tous les SMS reçus qui ont un identifiant univoque (champ id_message de la classe SMS_MO) supérieur à celui passé comme paramètre; les identifiants sont des numéros entiers plus grands que zéro; donc, en passant zéro comme paramètre, vous obtenez en réponse tous les SMS reçus par l'utilisateur.

Exemple :

```
try {
    List<SMS_MO> smss_mo = connection.getNewSMS_MOs();
    if (smss_mo.isEmpty()) {
            System.out.println("no new message received");
    } else {
            for (SMS_MO sms_mo : smss_mo) {
                    System.out.println("New SMS id "+sms_mo.getId_message()+":");
                    System.out.println("  - reception service num. "+sms_mo.getSms_recipient());
                    System.out.println("  - sender: "+sms_mo.getSms_sender());
                    System.out.println("  - text SMS: "+sms_mo.getMessage());
                    System.out.println("  - sent "+sms_mo.getSend_date());
            }
    }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

GESTION DES SOUS-COMPTES

Les utilisateurs qui disposent d'un super-compte peuvent utiliser une des méthodes de l'objet SMSConnection pour monitorer et gérer ses propres sous-comptes. Plus précisément:

- getSubaccounts(), renvoie une List<Subaccount> contenant tous les sous-comptes créés
- createSubaccount(Subaccount) permet de créer un nouveau sous-compte; le login et le password sont générés par mobyt et doivent être lus par l'objet Subaccount qui est renvoyé
- lockSubaccount(Subaccount) et unlockSubaccount(Subaccount), permettent d'activer/desactiver un sous-compte; dans les deux cas un Subaccount est restitué.

Exemple :

```
Try {
    List<Subaccount> subs = connection.getSubaccounts();
    if (subs.isEmpty()) {
            System.out.println("no subaccount created");
    } else {
            for (Subaccount subacc: subs) {
                    System.out.println("Subaccount:");
                    System.out.println("Login "+subacc.getLogin());
                    System.out.println("Password "+subacc.getPassword());
                    System.out.println("Locked "+ (subacc.isActive ? "no" : "yes" ));
            }
    }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

GESTION DU CRÉDIT DES SOUS-COMPTES

Les utilisateurs qui disposent d'un super-compte peuvent attribuer/enlever du crédit à leurs propres sous-comptes.
Différentes méthodes de gestion du crédit SMS sont disponibles en fonction des caractéristiques du sous-compte.

- Si le paramètre de gestion des crédits est USE_SUPER_CREDIT il n'est pas possible d'attribuer/enlever du crédit au sous-compte.
- Si le sous-compte HAS_CREDIT il est possible de lui attribuer/enlever des crédits moveCredits(CreditMovement). Pour connaître le crédit du sous-compte, utiliser la méthode getSubaccountCredits(Subaccount), qui renvoie une liste Credit.
- Si le sous-compte utilise le paramètre USE_BOTH_CREDITS il est possible de lui attribuer du crédit en utilisant la méthode createPurchase(CreditMovement) et de lui l'enlever avec la méthode deletePurchase(CreditMovement). Pour visualiser les transfèrts de crédits depuis le super-compte vers les sous-comptes, utiliser la méthode getPurchases(Subaccount), qui renvoie une List<CreditMovement>.

Exemple :

```
try {
    Subaccount sub1 = connection.getSubaccounts().get(0);
    if (sub1.getCredit_mode() == Subaccount.HAS_CREDIT) {
//No movements allowed here!
    }
    else if (sub1.getCredit_mode() == Subaccount.USE_SUPER_CREDIT) {
//Getting available credits
            List<Credit> subCredits = connection.getSubaccountCredits(sub1);
            if (subCredits.size()>0) {
                    System.out.println("Nation " +subCredits.get(0).getNation().getIso3166() + " ("
                    + subCredits.get(0).getCreditType() + "): " + subCredits.get(0).getCount() );
            }
//Giving some credit...
            CreditMovement newmovement = new CreditMovement();
            newmovement.setSubaccount_login(sub1.getLogin());
            newmovement.setSuper_to_sub(true);
            newmovement.setSms_type(SMSType.TOP);
            newmovement.setAmount(100);
            connection.moveCredits(newmovement);
            }
    else if (sub1.getCredit_mode() == Subaccount.USE_BOTH_CREDITS) {
//Getting purchases list
            List<CreditMovement> purchases = connection.getPurchases(sub1);
//Deleting first purchase
            if (purchases.size()>0) {
                    connection.deletePurchase(purchases.get(0));
```

```
            }
//Creating new purchase
            CreditMovement newpurch = new CreditMovement();
            newpurch.setSubaccount_login(sub1.getLogin());
            newpurch.setSms_types(new SMSType[]{SMSType.TOP, SMSType.GOLD});
            newpurch.setPricePerMessage(new double[]{0.010, 0.009});
            newpurch.setPrice(50.00);
            connection.createPurchase(newpurch);
    }
} catch (SMSCRemoteException smscre) {
    System.out.println("Exception from mobyt server: "+smscre.getMessage());
}
```

TELECHARGEMENT DE LA LIBRAIRIE

Vous trouverez l'archive contenant les fichiers source du SDK à cette adresse :
http://www.mobyt.fr/pdf/sdk/sms_gateway_java.zip