PLATAFORMA DE DESENVOLVEDORES vivo

Manual da API SMS REST

versão 1.0

Conteúdo

1 INTRODUÇÃO

Este documento serve como guia inicial para que desenvolvedores utilizem a API REST que é responsável pelo envio de SMS da Plataforma de Desenvolvedores Vivo.

As funcionalidades da API são o envio de SMS e a consulta do estado de envio de um SMS. Estas funcionalidades são expostas através de uma interface REST (REpresentational State Transfer) que expõe o serviço simplificando o uso por meio de simples requisições HTTP.

1.1 ALCANCE

A API SMS da Plataforma de Desenvolvedores Vivo permite o envio de mensagens SMS somente para o BRASIL.

1.2 GLOSSÁRIO

- API: Application Programming Interface
- ID: Identifier
- HTTPS: HyperText Transfer Protocol Secure
- JSON: JavaScript Object Notation
- REST: Representational State Transfer
- SMS: Short Messaging Service
- URI: Uniform Resource Identifier
- URL: Uniform Resource Locator
- WSDL: Web Services Description Language

2 CONVENÇÕES GERAIS

2.1 CONSIDERAÇÕES GERAIS DE UMA INTERFACE REST

O REST (REpresentational States Transfer) é um estilo de arquitetura baseado nos seguintes princípios:

- **Direcionabilidade**: Os recursos são expostos através de URIs.
- **Sem Estado:** As requisições de recursos são independentes uma das outras.
- **Conectividade**: Os recursos podem incluir links para outros recursos.
- **Uma interface uniforme:** As operações permitidas são a obtenção, criação, alteração e eliminação de recursos utilizando o protocolo HTTP.

A implementação destes pilares tem como resultado serviços RESTful baseados no protocolo HTTP que são independentes da linguagem podendo ser utilizados perante a presença de firewalls, onde as aplicações podem cacheá-las, sendo elas altamente escaláveis, etc.

O REST tem como propósito a implementação de serviços leves, inteligíveis e facilmente implementáveis que se definem com base numa série de operações RESTful, que implica a troca de informações de acordo com os formatos de dados REST.

2.2 CONSIDERAÇÕES ESPECÍFICAS PARA A API REST DE SMS

A requisição de envio de SMS é uma requisição POST que se pode fazer com os seguintes Content-Types:

- *application/xml*
- *application/json*
- *application/x-www-form-urlencoded* (suportado, apesar de ser recomendado usar XML ou JSON)

2. A resposta a essa requisição irá no formato expresso na mesma, exceto quando a requisição seja url-encoded, no qual as respostas serão XML.

3. A requisição do SMSDeliveryStatus é uma requisição GET, na qual se indica o SMS a consultar através de um parâmetro indicado na URI que forma a requisição. Por padrão, a resposta vem em formato XML, mesmo que se possa pedir que a resposta esteja em formato JSON através de um parâmetro “alt”, incluído na URI que elabora a requisição.

4. **Mapeamento de XML para JSON**. Dado que o XML é o formato padrão utilizado na API de SMS, se incluem arquivos XSD que descrevem os dados necessários para invocar a API através de XML. Para passar estas representações para o formato JSON, apresentam-se as seguintes regras de aplicação geral:

    a. Os elementos XML que aparecem no mesmo nível hierárquico XML (tanto os elementos de primeiro nível como os que estão dentro do mesmo elemento XML pai), são mapeados para um conjunto de pares nome:valor dentro de um objeto JSON, como se descreve a seguir:

i. Cada elemento XML que aparece apenas uma vez no mesmo nível hierárquico é mapeado para um par *nome:valor* individual. O nome é formado de acordo com o ponto b, enquanto o valor é formado de acordo com o ponto c.

ii. Elementos XML que apareçam mais de uma vez no mesmo nível hierárquico são mapeados para um único par *nome:valor* individual. O nome é formado de acordo com o ponto b, enquanto o valor é um JSON que contém um valor por cada ocorrência do elemento XML. O nome é formado de acordo com o ponto b, enquanto os valores são formados de acordo com o ponto c

iii. O nome e o valor dos objetos JSON irão entre aspas "". Além disso, qualquer representação JSON irá entre chaves {}, em concordância com a RFC de JSON.

b. O nome do par *nome:valor* é o nome dos elementos XML (nome_elemento_XML:valor). O valor é formado como se descreve a seguir:

i. Quando o elemento XML não tem nem atributos nem elementos XML filhos, o valor é igual ao valor do elemento XML. No caso em que o elemento seja nulo (não tenha valor), se indicará pondo um valor “null” no JSON.

ii. Quando o elemento XML tiver elementos-filho e/ou atributos, o *valor* é um objeto JSON que contém os seguintes pares *nome:valor*:

- Um par *nome:valor* para cada atributo, onde *nome* é o nome do atributo e *valor* é o valor do atributo.

- Um par *nome:valor* associado ao valor do elemento XML, onde *nome* é a string “$t” e *valor* é o valor do elemento XML.

  Nota: não existe uma regra específica sobre isto na RFC de JSON ou em json.org. Portanto, se selecionou a string “$t” tendo como base as regras do Google para conversão de feeds XML para JSON

  (http://code.google.com/intl/es/apis/gdata/json.html).

- Pares *nome:valor* associados a elementos XML filhos.Estes pares *nome:valor* se formam de acordo com o ponto a. Dentro de JSON não é necessário refletir:

  i. O primeiro tag <?xml version="1.0" encoding="UTF-8" ?>
  ii. Declaração de namespaces ou schemaLocations.

Exemplo de transformação de XML a JSON.

```
<Animals>
   <dog>

      <name attr="1234">Rufus</name>
      <breed>labrador</breed>
```

```
   </dog>
   <dog>
      <name>Marty</name>
      <breed>whippet</breed>
       <a/>
   </dog>
   <dog/>
   <cat name="Matilda"/>
   <a/>
</Animals>

{"Animals": {
   "a": null,
   "cat": {"name": "Matilda"},
   "dog": [

      {
         "breed": "labrador",
         "name": {
            "$t": "Rufus",
            "attr": "1234"
         }
      },
      {
         "breed": "whippet",
         "a": null,
         "name": "Marty"
      },
      null
   ]
}}
```

2.2.1 Considerações de Segurança

O seguinte ponto detalha as considerações de segurança que devem ser levadas em conta nos aplicativos desenvolvidos para acessar à API de SMS REST.

Os aplicativos devem incluir um cabeçalho de autenticação em cada requisição HTTP. Este cabeçalho conterá os seguintes dados:

```
Authorization: SDPBasicAuth realm="SDPAPIs", consumer_key=
"serviceId@spId", signature_method="MD5",
signature="MD5(spId+spPassword+timeStamp)",
timestamp="YYYYMMDDHHMMSS", version="0.1", token="AccessToken",
requestor_id="MSISDN ", requestor_type="1"
```

Os termos **serviceId**, **spId** e **spPassword** se obtêm na página de informação do desenvolvedor no portal da Plataforma de Desenvolvedores Vivo.

O **requestor_id** é o MSISDN (número de telefone) do usuário em nome do qual o aplicativo invocará a API.

O **AccessToken** é um conjunto de 8 carateres dentre os seguintes {a-z, A-Z, 0-9} que o usuário poderá obter e atualizar na seção “Perfil -> Botão Editar -> Informações Pessoais -> Gerar Token” do portal da Plataforma de Desenvolvedores Vivo e que o aplicativo deve incluir no cabeçalho de autenticação para invocar a API em nome do usuário.

Dado que o aplicativo deve incluir o **requestor_id** e **token** do usuário, é de responsabilidade do desenvolvedor solicitar estes dados ao usuário na forma que considere mais conveniente antes de invocar a API.

Um exemplo do cabeçalho de autenticação segue abaixo:

```
Authorization: SDPBasicAuth realm="SDPAPIs", consumer_key=
"35000001000001@35000001", signature_method="MD5",
signature="5b941f30e158d2af2df658dd5acca810",
timestamp="20100127120206", version="0.1", token="1x3rs6n8",
requestor_id="551199999999", requestor_type="1"
```

Quando uma requisição HTTP é realizada sem o uso do cabeçalho ou com um formato incorreto, o aplicativo receberá uma resposta de erro HTTP. No entanto, se o formato do cabeçalho de autenticação estiver correto mas os valores errados, a requisição deverá ser aceita, mas a mensagem não será enviada. Esta situação poderá ser comprovada por meio da operação GetSMSDeliveryStatus (ver 3.2 CONSULTA DO ESTADO DE ENVIO).

3 DEFINIÇÃO DAS OPERAÇÕES

Neste capítulo se descrevem as operações de envio e consulta de recepção de mensagens. Através da API não só se podem enviar SMS como também se pode consultar o resultado da requisição.

As operações disponíveis com os tipos de dados de entrada e saída estão descritos no seguinte quadro.

Operação	Entrada	Saída
SendSMS	SMSTextType	SMSTextResultType
GetSMS DeliveryStatus	SMSDeliveryStatusPollType	SMSDeliveryStatusType

3.1 ENVIO DE SMS

3.1.1 Requisição

Operação	Entrada
HTTP Method	POST
URL	https://sdp.vivo.com.br/osg/UNICA-SMS-REST/SMS
Content	Os formatos de codificação são os seguintes: application/xml application/json application/x-www-form-urlencoded

Alguns exemplos de requisições com diferentes Content-Type:

Content-Type: application/json

```
POST /osg/UNICA-SMS-REST/SMS HTTP/1.1
Content-Type: application/json
Authorization: SDPBasicAuth realm="SDPAPIs",
consumer_key="0100105600@000025", signature_method="MD5",
signature="A5766BB000AEA947429B07D8F6896019",
timestamp="20100127120206", version="0.1", token="2SDO4gAL",
requestor_id="551199995555", requestor_type="1"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: sdp.vivo.com.br
Content-Length: 155

{"smsText": {
    "address": {"phoneNumber": "551199997777"},
    "message": "Este e o conteudo da mensagem."
}}
```

Content-Type: application/x-www-form-urlencoded

```
POST /osg/UNICA-SMS-REST/SMS HTTP/1.1
Content-Type: application/x-www-form-urlencoded
```

```
Authorization: SDPBasicAuth realm="SDPAPIs",
consumer_key="01001056000@000025", signature_method="MD5",
signature="D645CDAD8BFE153CCE0DC368970F8042",
timestamp="20100127120206", version="0.1", token="2SDO4gAL",
requestor_id="551199995555", requestor_type="1"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: sdp.vivo.com.br
Content-Length: 114

address.phoneNumber=551199997777&message=Este+e+o+conteudo+da+mensagem
```

Content-Type: application/xml

```
POST /osg/UNICA-SMS-REST/SMS HTTP/1.1
Content-Type: application/xml
Authorization: SDPBasicAuth realm="SDPAPIs",
consumer_key="01001056000@000025", signature_method="MD5",
signature="CF753CB3173FC8F343C2B84EFEDE3AED",
timestamp="20100127120206", version="0.1", token="2SDO4gAL",
requestor_id="551199995555", requestor_type="1"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: sdp.vivo.com.br
Content-Length: 497

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ns0:smsText
xmlns:ns0="http://www.telefonica.com/UNICA/sms_bodies/v0_1_1_LITMUS_LA
TAM_03102009"
xmlns:ns1="http://www.telefonica.com/UNICA/sms_types/v0_1_1_LITMUS_LAT
AM_03102009"
xmlns:ns2="http://www.telefonica.com/UNICA_CommonTypes/v0_1">
  <ns1:address>
    <ns2:phoneNumber>551199997777</ns2:phoneNumber>
  </ns1:address>
  <ns1:message>Este é o conteúdo da mensagem.</ns1:message>
</ns0:smsText>
```

3.1.2 RESPOSTA

Operação	Entrada
HTTP Response Code	201 Created
Content-Type Headers	application/json, caso a requisição tenha sido feita em JSON application/xml, caso a requisição tenha sido feita em XML ou em URLencoded
Location Header	Contém a URL que permite realizar a operação de consulta do estado de envio da mensagem
Body	Elemento SMSTextResultType no formato indicado no cabeçalho Content-Type

Caso ocorra algum erro relacionado com:

- spId ou serviceId inválidos
- spPassword inválido
- O aplicativo não tem permissões para utilizar a API

- Parâmetros inválidos
- Violação dos SLA

Será respondido com um código de estado HTTP do tipo 4XX/5XX.

No entanto, se ocorre um erro relacionado com:

- token inválido
- O MSISDN não tem permissões para utilizar a API

Será respondido com um código de estado HTTP 201, mas ao consultar o estado do envio será obtido o valor DeliveryImpossible (ver 3.2 CONSULTA DO ESTADO DE ENVIO).

Alguns exemplos de respostas com diferentes Content-Type:

Content-Type: application/json

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: https://sdp.vivo.com.br/osg/UNICA-SMS-
REST/SMS/SMSDeliveryStatus?smsIdentifier=10000910151003485403
Content-Type: application/json;charset=ISO-8859-1
Content-Length: 51
Date: Thu, 15 Oct 2009 10:03:48 GMT

{"smsTextResult":{"result":"100009101510034854O3"}}
```

Content-Type: application/xml

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: https://sdp.vivo.com.br/osg/UNICA-SMS-
REST/SMS/SMSDeliveryStatus?smsIdentifier=10000910151000384995
Content-Type: application/xml;charset=ISO-8859-1
Content-Length: 240
Date: Thu, 15 Oct 2009 10:00:38 GMT

<smsTextResult
xmlns="http://www.telefonica.com/UNICA/sms_bodies/v0_1_1_LITMUS_LATAM_
03102009"
xmlns:v0="http://www.telefonica.com/UNICA/sms_types/v0_1_1_LITMUS_LATA
M_03102009">
  <v0:result>10000910151000384995</v0:result>
</smsTextResult>
```

3.2 CONSULTA DO ESTADO DE ENVIO

3.2.1 REQUISIÇÃO

A URL necessária para a requisição de consulta se obtém no cabeçalho incluído na resposta à operação SendSMS.

Operação	Entrada
HTTP Method	GET
URL	https://sdp.vivo.com.br/osg/UNICA-SMS-REST/SMSDeliveryStatus
Body	A requisição GET de consulta não inclui body, a informação necessária deve ser passada como parâmetros de consulta.

Exemplo de uma requisição de consulta do Estado de Envio com resposta em XML

```
GET /osg/UNICA-SMS-
REST/SMSDeliveryStatus?smsIdentifier=10000910151005185603 HTTP/1.1
Authorization: SDPBasicAuth realm="SDPAPIs",
consumer_key="0100105600@000025", signature_method="MD5",
signature="4E787BC6A43561968224BD9ACDFCBE3A",
timestamp="20100127120206", version="0.1", token="2SDO4gAL",
requestor_id="551199995555", requestor_type="1"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: sdp.vivo.com.br
```

Exemplo de uma requisição de consulta do Estado de Envio com resposta em JSON

```
GET /osg/UNICA-SMS-
REST/SMSDeliveryStatus?smsIdentifier=100009101510140464l4&alt=json
HTTP/1.1
Authorization: SDPBasicAuth realm="SDPAPIs",
consumer_key="0100105600@000025", signature_method="MD5",
signature="2760345011F7C1B49BEA572EEE235F2A",
timestamp="20100127120206", version="0.1", token="2SDO4gAL",
requestor_id="551199995555", requestor_type="1"
User-Agent: Jakarta Commons-HttpClient/3.1
Host: sdp.vivo.com.br
```

3.2.2 Resposta

Operação	Entrada
HTTP Response Code	200 OK
Content-Type Headers	application/json application/xml (padrão)
Body	Um elemento do tipo SMSDeliveryStatusType indicado no formato expresso no Content-Type

Exemplo de uma resposta em JSON

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=ISO-8859-1
Content-Length: 126
Date: Thu, 15 Oct 2009 10:14:05 GMT

{"smsDeliveryStatus":{"smsDeliveryStatus":{"address":{"anyUri":"tel:+
551199997777"},"deliveryStatus":"DeliveredToTerminal"}}}
```

Exemplo de uma resposta em XML

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=ISO-8859-1
Content-Length: 481
Date: Thu, 15 Oct 2009 10:05:19 GMT

<NS1:smsDeliveryStatus
xmlns:NS1="http://www.telefonica.com/UNICA/sms_bodies/v0_1_1_LITMUS_LA
TAM_03102009">
  <NS2:smsDeliveryStatus
xmlns:NS2="http://www.telefonica.com/UNICA/sms_types/v0_1_1_LITMUS_LAT
AM_03102009">
    <NS2:address>
      <NS3:anyUri
xmlns:NS3="http://www.telefonica.com/UNICA_CommonTypes/v0_1">tel:+5511
99999999</NS3:anyUri>
    </NS2:address>
    <NS2:deliveryStatus>DeliveredToTerminal</NS2:deliveryStatus>
  </NS2:smsDeliveryStatus>
</NS1:smsDeliveryStatus>
```

4 NAMESPACES

Os tipos de dados vêm definidos no seguinte namespace:
http://www.telefonica.com/UNICA/sms_types/v0_1_1_LITMUS_LATAM_03102009

Os elementos superiores utilizados nos corpos das requisições e respostas REST vêm definidos no seguinte namespace:
http://www.telefonica.com/UNICA/sms_bodies/v0_1_1_LITMUS_LATAM_03102009

Os tipos de dados comuns para todas as APIs vêm definidos no seguinte namespace:
http://www.telefonica.com/UNICA_CommonTypes/v0_1

Outros tipos de dados comuns adotados de ParlayX vêm definidos no seguinte namespace:
http://www.csapi.org/schema/parlayx/common/v3_1

5 DEFINIÇÃO DOS TIPOS DE DADOS

Neste capítulo mostram-se definições dos tipos de dados utilizados por esta API. As seguintes considerações devem ser tidas em conta:

- As definições são mapeamentos diretos para XML Schemas
- A definição destes tipos de dados está baseada em GSMA OneAPI [3] e ParlayX 3.1 [2]. Em alguns casos, o tipo de dado corresponde diretamente a um tipo de dado definido em ParlayX. Portanto, nos esquemas proporcionados importam-se tipos de dados de ParlayX.

5.1 ESTRUTURA DO SMSTextType

Informação para o envio de um SMS.

Parâmetro	Elemento tipo	Opcional	Descrição
Address	uct:UserIdType [1..unbounded]	Não	Lista de endereços para os quais se enviarão os SMS. IdToken e valores de IP não são aceitos.
Message	xsd: string	Não	Mensagem a ser enviada. Tamanho máximo 160 carateres (leve em conta que os carateres com acentos ocupam dois carateres quando são enviados). Em alguns casos, os carateres com acentos podem não ser visualizados corretamente no terminal.
Encode	xsd:string	Sim	Quando a codificação é de base64, esta operação permite enviar um SMS de aviso ou push SMS, com os seguintes parâmetros extensíveis: *sourceport, destinationport, esm_class* e *data_coding.*
Sourceport	xsd:int	Sim	Indica o número da porta da aplicação associado ao endereço de origem da mensagem. Opcional para *MyMail* service.
destinationport	xsd:int	Sim	Indica o número da porta da aplicação associado ao endereço de destino da mensagem.
esm_class	xsd:int	Sim	Este parâmetro é utilizado para indicar atributos especiais com uma mensagem curta.
data_coding	xsd:int	Sim	Define a codificação dos dados de usuário da mensagem curta.

5.2 ESTRUTURA DO SMSTextResultType

Resposta da operação REST SendSMS para consulta do estado de entrega.

Parâmetro	Elemento tipo	Opcional	Descrição
Result	xsd: string	Não	É um identificador correspondente com o SMS enviado utilizado numa operação GetSMSDeliveryStatus

5.3 ESTRUTURA DO SMSDeliveryStatusPollType

Utilizado como entrada para a consulta ou recepção síncrôna do estado de entrega de um SMS.

Deve-se ter em conta que o tipo de dado *SMSDeliveryStatusPollType* se utiliza nas operações *GET* e, portanto, deve ser passado na URI de requisição. Desta maneira, não se utiliza XML nem JSON para a codificação deste tipo de dados.

Parâmetro	Elemento tipo	Opcional	Descrição
smsIdentifier	xsd: string	Não	Identificador relacionado com o estado de entrega do SMS.
Alt	uct:AltType	Sim	Parâmetro opcional que serve para indicar que a consulta se realiza em formato JSON. (alt=JSON)

5.4 ESTRUTURA DO SMSDeliveryStatusType

Informação do estado de entrega do SMS.

Nota: Utiliza-se como base GSMA OneAPI [3] e ParlayX 3.1 [2].

Parâmetro	Elemento tipo	Opcional	Descrição
smsDeliveryStatus	DeliveryInformationType [0..unbounded]	Sim	Descreve as variações do estado de entrega do SMS. Os valores possíveis são: • DeliveredToNetwork • DeliveryUncertain • DeliveryImpossible • MessageWaiting • DeliveredToTerminal

5.5 ESTRUTURA DO DeliveryInformationType

Informação da entrega do SMS

Parâmetro	Elemento tipo	Opcional	Descrição
Address	uct:UserIdType	Não	Indica o endereço de destino relacionado com a notificação.
deliveryStatus	DeliveryStatus Type	Não	Indica o resultado da entrega para os endereços de destino.
description	xsd:string	Sim	Utilizado juntamente com o estado de entrega (ex:DeliveryImpossible) para proporcionar informação adicional.

Nota: Tirados de ParlayX 3.1 [2]

5.6 ENUMERAÇÃO DO DeliveryStatusType

Lista de valores possível do estado de Entrega.

Enumeração	Descrição
DeliveredToNetwork	Entrega à rede com sucesso
DeliveryUncertain	Estado de entrega desconhecido. (ex: por ter sido enviado para outra rede
DeliveryImpossible	Entrega impossível, a mensagem não pôde ser entregue antes de expirar
MessageWaiting	A mensagem está em fila de espera para ser entregue. Este é um estado temporário, pendente de transição para outro estado mencionado.
DeliveredToTerminal	Entrega bem-sucedida
DeliveryNotificationNotSupported	Impossível fornecer a notificação de entrega. Utilizado para indicar que a recepção de entrega para o endereço especificado numa operação *SendSMS* não está suportada.

Nota: tirados de ParlayX 3.1 [2]

5.7 ENUMERAÇÃO DO AltType

Parâmetro para solicitar uma resposta em formato JSON.

Enumeração	Descrição
JSON	O formato do conteúdo da resposta deve ser JSON

5.8 OPÇÃO UserIdType

Parâmetro	Elemento tipo	Opcional	Descrição
phoneNumber	E164Type	Sim	Número de telefone
anyUri	xsd:anyURI	Sim	Qualquer URI
ipAddress	IpAddressType	Sim	Endereço IP
alias	AliasType	Sim	Alias
otherId	OtherIdType	Sim	Qualquer outro tipo de identidade de usuário

6 BIBLIOTECAS DE USO DAS APIS

6.1 CLIENTE JAVA

6.1.1 Diretrizes de Programação

Geralmente é definida uma *interface* principal para cada API, que define as operações REST do cliente permitidas para o serviço correspondente. Através de um mecanismo de *factories* se oferecem implementações destas *interfaces*, que funcionam como clientes REST. Estes clientes implementam operações definidas para cada API utilizando um modelo de dados Java de acordo com os tipos de dados definidos nas APIs.

A *interface* implementada pelos clientes SMS é a seguinte:

- SMS API Client: es.tid.unica.rest.sms.SMSClient

A *factory* que deve ser usada para obter uma instância da classe cliente para SMS é a seguinte:

- SMS API Client Factory: es.tid.unica.rest.sms.SMSClientFactory

6.1.2 Exemplo para o envio com o cliente SMS

O exemplo a seguir mostra o uso do cliente para o uso da API REST de SMS:

- Em primeiro lugar se realiza a construção do objeto cliente (SMSClient), mediante o uso de uma *factory* (SMSClientFactory)
- A seguir se configura o cliente com os dados necessários, tanto para o estabelecimento da conexão com o servidor (incluindo a utilização de um proxy, se necessário, assim como o *endpoint* no qual se encontra o serviço ao que se vai invocar), como para permitir a autenticação do cliente. Para tanto é preciso especificar os elementos que formam as credenciais de segurança que serão utilizadas.
- Uma vez feitas as configurações, já é possível realizar o envio de um SMS, utilizando o método *sendSMS*. Este método recebe como parâmetro a mensagem construída utilizando as classes proporcionadas pela API, que seguem exatamente as especificações de tipos de dados explicados nas seções anteriores, assim como a codificação empregada.
- Uma vez enviado o SMS se obtém como resposta um objeto, da classe *SMSResult*, que contém o resultado da operação de envio, incluindo a URI que poderá ser utilizada para consultar o *delivery status* do SMS.

```
import es.tid.unica.rest.Encoding;
import es.tid.unica.rest.sms.SMSClient;
import es.tid.unica.rest.sms.SMSClientFactory;
import es.tid.unica.rest.sms.SendSMSResult;
import es.tid.unica.types.common.E164Type;
import es.tid.unica.types.common.UserIdType;
import es.tid.unica.types.sms.SMSDeliveryStatusType;
import es.tid.unica.types.sms.SMSTextContentType;
import es.tid.unica.types.sms.SMSTextType;


// Configuração geral de dados
String base_uri = "https://sdp.vivo.com.br/osg/UNICA-SMS-REST";
String proxy_host = null;  // No proxy is used
int proxy_port = 80;

// Configuração de dados de segurança
String service_id = "0100105600";
String sp_id = "000025";
String sp_password = "123456";
```

```
String access_token = "2SDO4gAL";
String requestor_id = " 551199995555";
int requestor type = 1;

//
String receipt_msisdn = " 551199997777";
String message_text = "Este e o texto da mensagem";

try {
        // Criando o cliente
        SMSClient client = SMSClientFactory.getInstance().createSMSClient(base_uri,
proxy_host, proxy_port);

        // Inicializando o cliente
        client.setServiceId(service id);
        client.setSpId(sp_id);
        client.setSpPassword(sp_password);
        client.setAccessToken(access_token);
        client.setRequestorId(requestor_id);
        client.setRequestorType(requestor type);

        // Autenticação Habilitada
        client.enableAuthentication(true);

        //-----------------------------------------------
        // Envio de um SMS
        // 1. Construir a mensagem
        SMSTextType sms = new SMSTextType();
        // 1.1 Receber endereço
        UserIdType address = new UserIdType();
        address.setPhoneNumber(new E164Type(receipt msisdn));
        sms.addAddress(address);
        // 1.2 Conteudo do texto
        SMSTextContentType sms_text_content = new SMSTextContentType(message_text);
        sms.setMessage(sms_text_content);
        // 2. Ebvio da mensagem
        SendSMSResult result = client.sendSMS(sms, Encoding.APPLICATION XML);
        System.out.println("O resultado de sendSMS opeation é " + result);
        //-----------------------------------------------

        //-----------------------------------------------
        // Recuperação do status de entrega do SMS enviado
        SMSDeliveryStatusType status =
client.getSMSDeliveryStatus(result.getLocationHeader().toString(), null);
        System.out.println("O resultado de getSMSDeliveryStatus é " + status);
        //-----------------------------------------------

} catch (Exception e) {
        e.printStackTrace();
}
```

6.1.3 Pacotes do Cliente

A biblioteca Java de cliente SMS é formada pelos seguintes pacotes:

Enumeração	Descrição
es.tid.unica.rest.sms	Pacote que contém as classes principais para o cliente SMS REST API.
es.tid.unica.types.common	Pacote que contém os tipos de dados principais para as mensagens e os parâmetros utilizados pelas APIs
es.tid.unica.types.parlayx.common	Este pacote contém os tipos de dados básicos de ParlayX, que podem ser utilizados nas APIs

Enumeração	Descrição
es.tid.unica.types.sms	Pacote que contém os tipos de dados específicos da API de SMS

6.1.4 Pré-requisitos

Para o correto funcionamento da biblioteca de cliente de Java de SMS é necessário ter instalada a versão 1.5 ou superior do JRE (Java Runtime Environment) ou JDK (Java Development Kit).

Também é necessário utilizar as bibliotecas contidas nos arquivos .jar situados no diretório *lib* da distribuição do cliente Java.

6.2 CLIENTE C#

6.2.1 Diretrizes de Programação

Define-se uma classe *factory* para cada API atuando como criador de clientes REST. A partir de uma instância única da *factory* obtém-se um cliente para a API ao que pertence, povoado com mais ou menos dados iniciais dependendo da operação de criação utilizada. O cliente obtido, por flexibilidade na hora de fornecer diferentes implementações, é oferecido em forma de interface da linguagem C#.

A interface obtida a partir do *factory* se comportará como um cliente REST. Estes clientes implementam as operações definidas para cada API utilizando um modelo de dados C# de acordo com os tipos de dados definidos nas APIs.

A *factory* e a interface do cliente SMS para a API REST são os seguintes:

- SMS Client Factory: es.tid.unica.rest.sms.SMSClientFactory
- SMS Client Interface API: es.tid.unica.rest.sms.ISMSClient

6.2.2 Exemplo para o envio com o cliente SMS

O exemplo seguinte mostra o uso do cliente para o uso da API REST de SMS:

```
try
  {
    //1.- Instanciar factory e crier cliente da instancia de factory.
    //Neste exemplo usaremos baseUri e dados de proxy
    ISMSClient smsClient = SMSClientFactory.GetInstance().CreateSMSClient(baseUri,
proxyHost, proxyPort);
    //2.- Setar credenciais para autenticação
    smsClient.Credentials.ServiceId = serviceId;
    smsClient.Credentials.SpId = spId;
    smsClient.Credentials.SpPassword = spPassword;
    smsClient.Credentials.RequestorId = accessNumber;
    smsClient.Credentials.RequestorType = requestorType;
    smsClient.Credentials.AccessToken = accessToken;
    //3.-Criar o objeto de  mensagem de texto
    SMSTextType sms = new SMSTextType();
    //4.- Criar pelo menos um endereço de destino e adicioná-lo no objeto da mensagem
    UserIdType address = new UserIdType();
```

```
    address.PhoneNumber = new E164Type("551199997777");
    sms.AddAddress(address);
    //5.- Criar o conteúdo da mensagem interna e adicioná-la no obejto da mensagem
    SMSTextContentType message = new SMSTextContentType("Este e o conteudo");
    sms.Message = message;
    //6.- Definindo encoding
    Encoding encoding = Encoding.APPLICATION_URL_ENCODED;
    //6.- Envio de mensagem usando SMS do cliente
    SendSMSResult sendSMSResult=smsClient.SendSMS(sms, encoding);
    //7.- Checar o identificador da mensagem de resultado de entrega
    Console.WriteLine("SMS Identifier: {0}", sendSMSResult.SmsTextResult.Result.Data);
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex.Message);
   }
```

6.2.3 Pacotes do Cliente

Os pacotes são descritos a seguir:

Enumeração	Descrição
es.tid.unica.rest.sms	Pacote que contém as classes principais para o cliente SMS REST API.
es.tid.unica.types.common	Pacote que contém os tipos de dados principais para as mensagens e os parâmetros utilizados pelas APIs..
es.tid.unica.types.parlayx.common	Este pacote contém os tipos de dados básicos de ParlayX, que podem ser utilizados nas APIs
es.tid.unica.types.sms	Pacote que contém os tipos de dados específicos da API de SMS

6.2.4 Pré-requisitos

Para o funcionamento correto das APIs C# são necessários os seguintes pré-requisitos:

- Sistema Operacional Windows XP Service Pack 2 (ou superior) ou Windows Vista

- Microsoft .NET Framework 3.5, disponível para baixar gratuitamente em http://www.microsoft.com/downloads

Esta biblioteca não foi concebida para outros ambientes como o Mono ou Portable.NET.

6.3 CLIENTE PHP

6.3.1 Dependências

- PHP ≥ 5.2.6
- PHP5-CURL ≥ 5.2.6

- CURL ≥ 7.18.2

6.3.2 Diretrizes de Programação

É definida uma classe para cada API atuando como um cliente REST. Estes clientes implementam as operações definidas para cada API utilizando como referência os tipos de dados definidos nas APIs.
As classes do cliente SMS para a API REST são as que seguem (uma classe para cada tipo de codificação):

- smsRESTlibraryXML.php
- smsRESTlibraryJSON.php
- smsRESTlibraryURLEnc.php

Esta biblioteca foi testada nos seguintes sistemas operacionais: CentOs 5, Fedora 11, Ubuntu Jaunty, Ubuntu Karmic, Debian Lenny, Windows 7 e Windows XP.

6.3.3 Exemplo de envio com o cliente SMS

O exemplo seguinte mostra o uso do cliente para a utilização da API REST de SMS:

```
<?php

//Classe para o envio de SMS por meio de URL encode
include_once "./smsRESTlibraryURLEnc.php";

//Criação do cliente para o envio de SMS
$smsClient = new
smsRESTclient($spId,$serviceId,$spPassword,$token,$requestor_id,$apiendpoint);

//Destino do SMS
$address["phoneNumber"]="0000000000";

//Conteudo do SMS
$message= "Esto es una prueba";

//Envio do SMS e o resultado
$result = $smsClient->sendSMS($address,$message);

?>
```

6.3.4 Pacotes do Cliente

Os pacotes são descritos a seguir:

Enumeração	Descrição
smsRESTlibraryXML.php	Pacote que contém a classe principal para o cliente SMS REST API com codificação XML
smsRESTlibraryJSON.php	Pacote que contém a classe principal para o cliente SMS REST API com codificação JSON.
smsRESTlibraryURLEnc.php	Pacote que contém a classe principal para o cliente SMS REST API com codificação URL encode.

7 DETALHE DAS DESCRIÇÕES DE ERRO

As operações descritas neste manual podem retornar códigos de erro HTTP, como se explica a seguir.

Caso ocorra algum erro relacionado com:

- spId ou serviceId inválidos
- spPassword inválido
- O aplicativo não tem permissões para utilizar a API
- Parâmetros inválidos
- Violação dos SLA

Será respondido com um código de estado HTTP do tipo 4XX/5XX.

No entanto, se ocorre um erro relacionado com:

- token inválido
- O MSISDN não tem permissões para utilizar a API

Será respondido com um código de estado HTTP 201, mas quando consultar o estado do envio irá obter o valor DeliveryImpossible

A. CONSIDERAÇÕES GERAIS

Nesta seção se descrevem elementos necessários para o funcionamento normal da API REST SMS:

- Métodos HTTP: que constituem as operações RESTful, de acordo com os princípios REST.
- Representações comuns.

A. 1. Métodos HTTP

As seguintes descrições foram extraídas da RFC de HTTP 1.1.

A.1.1. POST

Utiliza-se o método POST para pedir que o servidor de origem aceite a entidade contida na requisição como um novo recurso identificado por sua URI na Request-Line. O POST está concebido para permitir um método uniforme que englobe as seguintes funções:

- Anotação de recursos existentes.
- Postar uma mensagem num fórum, grupo de notícias, listas de correio, ou grupo semelhante de artigos.
- Fornecer um bloco de dados, como por exemplo o resultado do envio de um formulário, para que seja processado.
- Estender um banco de dados através do acréscimo de uma operação.

A funcionalidade executada pelo método POST é determinada pelo servidor e é dependente da Request-URI. A entidade postada está subordinada a essa URI da mesma maneira que o arquivo está subordinado ao diretório que o contém, que um artigo de notícias está subordinado a um grupo de notícias ao que está postado, como um registro está subordinado a um banco de dados.

A.1.2. GET

O método GET tenta extrair qualquer informação (na forma de uma entidade) identificada por uma Request-URI. Se a Request-URI se refere a um processo que gera dados, a resposta deve ser um dado produzido que se deve devolver como entidade na resposta e não o texto origem do processo, a não ser que o texto seja a saída do processo.

A.1.3. PUT

O método PUT solicita que a entidade contida seja armazenada abaixo da Request-URI proporcionada. Se a Request-URI se refere a um recurso existente, a entidade contida deve ser considerada como uma versão modificada do recurso residente no servidor. Se a Request-URI não aponta para um recurso existente, e a URI é capaz de ser definida como um novo recurso pelo agente solicitante, o servidor de origem pode criar o recurso com essa URI.

A.1.4. DELETE

O método Delete solicita que o servidor de origem elimine o recurso identificado com a Request-URI. Este método pode ser reescrito por intervenção humana (ou por outros métodos) no servidor de origem. O cliente não pode garantir que a operação tenha sido executada com sucesso, inclusive se o código de estado devolvido a partir da origem indica que a ação foi finalizada com sucesso. No entanto, o servidor não deve indicar uma resposta satisfatória a não ser que, no momento em que a resposta é entregue, tente eliminar o recurso ou movê-lo para uma localização inacessível.

A.2. REPRESENTAÇÕES COMUNS

A.2.1. JSON

As requisições POST podem incluir dados em formato JSON. As respostas devem incluir o corpo em formato JSON, se correspondem a requisições POST incluídas em formato JSON ou se correspondem a requisições GET incluídas com o parâmetro 'alt=json'. Nestes casos, deve estar presente na resposta o cabeçalho Content-Type:application/json.

A.2.2. XML

As requisições POST podem incluir dados em formato XML. Nestes casos é utilizado um corpo *application/xml*. Este formato XML deve cumprir com as especificações do XML Schema.
As respostas devem incluir um corpo do XML caso a correspondente requisição POST inclua dados em formato XML ou se a correspondente requisição GET não incluía o parâmetro 'alt=json'. Nestes casos, deve estar presente na resposta o cabeçalho *Content-Type: Application/xml*.

B. REFERÊNCIAS

[1] 3GPP TS 29.199-1: "Open Service Access (OSA); Parlay X Web Services; Part 1: Common"

[2] 3GPP TS 29.199-5: "Open Service Access (OSA); Parlay X Web Services; Part 5: Multimedia Messaging".

[3] GSMA OneAPI, http://gsma.securespsite.com/access/default.aspx

[4] RFC 2616: “Hypertext Transfer Protocol -- HTTP/1.1”

[5] W3C Recommendation (26 June 2007): "Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts”, https://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/#_http_binding_default_rule_method

[6] W3C Recommendation (2 May 2001): "XML Schema Part 2: Datatypes", http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.