MBS (Mobile Billing System)

User Guide

Version 1.5 (2012-03-07)

Content

Document management

Document metadata

Project title:	Mobile Billing System (MBS) – mobile users billing system		
Project leader:	Valdas Petrulis	**Document version:**	1.5
Authors:	Valdas Petrulis, Liutauras Ričkus	**Document version date:**	2012-03-07
Reviewer:	Valdas Petrulis	**Review data:**	2009-11-10

History of changes

Version	State	Date	Responsible person	Description
0.6	Changes	2009-11-09	Ina Bachova	Simplified WAP billing connection; Possibility to get newest subscriber status during WAP identification; Addition of security annex;
0.6	Review	2009-11-10	Valdas Petrulis	Review, cosmetic changes;
0.6	Document joining	2009-11-11	Valdas Petrulis	Addition of document management section;
0.7	Changes	2009-12-10	Ina Bachova	Simplified WAP subscription connection;
1.2	Addition	2010-08-10	Ina Bachova	Overal documention translated
1.2	Addition	2010-08-10	Valdas Petrulis	Service API – manage your services via SOAP
1.2	Addition	2010-09-15	Valdas Petrulis	Possibility to identify WAP user‘s payment type (prepaid, postpaid, testing, unknown)
1.3	Addition	2010-10-10	Valdas Petrulis	WEB users billing - SMS bank
1.3	Addition	2010-12-14	Mantas Litvaitis	Operator gateway IP list
1.3	Addition	2011-01-03	Mantas Litvaitis	New prices in Estonia
1.4	Addition	2011-12-07	Valdas Petrulis	SMS Mass Sending Services – possibility to send big SMS/WapPush packets
1.5	Addition	2012-03-07	Valdas Petrulis	„Omnitel Lietuva“ subscription renewal rules changed

Introduction

What is MBS?

Mobile Billing System (hereinafter – "MBS") is a system that allows the third parties to identify mobile users and charge them. When using MBS, the third party does not have to know how to recognize and charge every individual operator. This is done by MBS! Currently MBS is able to:

- Charge a user browsing WEB (SMS bank services)
- Charge users via short numbers (Keyword services)
- Periodically charge a user (Subscription Billing Services)
- Charge a user browsing WAP (WAP Billing Services)
- Identify a user browsing WAP (WAP Identification Services)
- Send information SMS/WapPush messages to users (SMS/WapPush Sending Services)

Target group of this document

This document is for developers willing to charge for their applications by using MBS. Some of the most popular cases where it is beneficial to connect to MBS:

- When using MBS, a user may be **charged** without leaving **WEB** browser by simply redirecting it to MBS! Thus for instance, one-off charge for virtual credits, money donating, etc.
- When using MBS a user may be **charged** by simply asking to send **SMS message to short number**. Thus for instance, one-off charge for user's profile publishing, virtual credits in WEB game, etc.
- When using MBS a user may be **charged periodically** for subscription, just ask user to send **SMS message to short number**. Thus for instance, VIP membership ordering and automatic extension in dating site, magazine, newspaper subscription, etc.
- When using MBS in your project, you can **identify** a unique user visiting your **WAP project**, its operator, language, country, even its phone number! With this information you can make your project more user-friendly and adjust it to an individual unique user.
- When using MBS, a user may be **charged** without leaving **WAP** browser by simply redirecting it to MBS! Thus for instance, one-off charge for melodies, backgrounds, etc.
- When using MBS, you can **register users to subscription** without leaving **WAP** browser! In this way you can make a membership in your project paid. Thus for instance, VIP members pay LTL 1 per week.
- When using MBS you can **inform** users by **SMS or WapPush messages**. Thus for instance, dating site sends reminders about profiles.

Service request and statistics

- **Service request** – manage your services at http://partners.vero.lt/, service statistics are also available here.
- Also, to request the service, please email us at info@vero.lt
- **Service API** - for wholesale partners, who wants to aggregate our services, we offer to use Service API, it allows to manage your services via SOAP.
- **Service staitistics** tools:
    - Keyword and subscription services: http://partners.vero.lt/
    - WAP services: http://my.vero.lt/
    - SMS/WapPush services: http://smscomm.vero.lt/
    - Promotional campaigns, lotteries: http://campaign.vero.lt/
- Payment reviews and subscription management for users: http://savitarna.vero.lt.

Technical references

- The latest MBS documentation, examples and the MBS PHP library are always located at http://www.veromobile.eu/en/support_centre;
- If you cannot find the answer to any technical question, please email us at support@vero.lt or contact via SKYPE (*Verosupport*);
- MBS operation demo (for testing): http://mbs.vero.lt/;
- WAP identification (for testing): http://bill.vero.lt/;
- WAP subscription registration server: http://bill.vero.lt/wap/register;
- WAP unsubscription registration server: http://bill.vero.lt/wap/unregister;
- WAP billing server: http://bill.vero.lt/wap/bill;
- WAP identification server: http://bill.vero.lt/identify;
- MBS WAP billing SOAP server: http://bill.vero.lt/soap/wap1.1, WSDL http://bill.vero.lt/soap/wap1.1?wsdl;

Supported countries and prices

Lithuania

Service	Supported	Comments
Short number charging	●	Short numbers: 1679, 1654, 1656
Periodical charging (subscriptions)	●	Short number 1679 and WAP
WAP charging	●	
WAP identification	●	
SMS/WapPush sending	●	
Advertising actions/loteries	●	

Price	SMS	WAP	Subscription
0.30 LTL	●	●	●
0.50 LTL	●	●	●
0.60 LTL	●	●	●
0.70 LTL	●	●	●
0.80 LTL	●	●	●
0.90 LTL	●	●	●
1.00 LTL	●	●	●
1.30 LTL	●	●	●
1.50 LTL	●	●	●
1.70 LTL	●	●	●
1.80 LTL	●	●	●
1.90 LTL	●	●	●
2.00 LTL	●	●	●
2.50 LTL	●	●	●
3.00 LTL	●	●	●
3.50 LTL	●	●	●
4.00 LTL	●	●	●
4.50 LTL	●	●	●
5.00 LTL	●	●	●
5.50 LTL	●	●	●
6.00 LTL	●	●	●
6.50 LTL	●	●	●
7.00 LTL	●	●	●
7.50 LTL	●	●	●

8.00 LTL	●	●	●
8.50 LTL	●	●	●
9.00 LTL	●	●	●
9.50 LTL	●	●	●
10.00 LTL	●	●	●
12.00 LTL	●	●	●
15.00 LTL	●	●	●

Schema 1 Supported prices in Lithuania

Latvia

Service	Supported	Comments
Short number charging	●	Short number 1850
Periodical charging (subscriptions)	●	Short number 1850 and WAP. LMT limits subscription services
WAP charging	●	
WAP identification	●	LMT does not identify MSISDN
SMS/WapPush sending	●	
Advertising actions/loteries	●	

Price	SMS	WAP	Subscription
0.05 LVL	●	●	●
0.10 LVL	●	●	●
0.15 LVL	●	●	●
0.20 LVL	●	●	●
0.25 LVL	●	●	●
0.30 LVL	●	●	●
0.35 LVL	●	●	●
0.40 LVL	●	●	●
0.45 LVL	●	●	●
0.50 LVL	●	●	●
0.55 LVL	●	●	●
0.60 LVL	●	●	●
0.65 LVL	●	●	●
0.70 LVL	●	●	●
0.75 LVL	●	●	●
0.80 LVL	●	●	●
0.85 LVL	●	●	●

0.90 LVL	●	●	●
0.95 LVL	●	●	●
1.00 LVL	●	●	●
1.25 LVL	●	●	●
1.50 LVL	●	●	●
1.75 LVL	●	●	●
2.00 LVL	●	●	●
2.25 LVL	●	●	●
2.50 LVL	●	●	●
2.75 LVL	●	●	●
3.00 LVL	●	●	●

Schema 2 Supported prices in Latvia

Estonia

Service	Supported	Comments
Short number charging	●	Short numbers: 15152, 1512, 13553, 15156, 15151, 15154, 15158
Periodical charging (subscriptions)	●	All operators restricts subscription services
WAP charging	●	
WAP identification	●	
SMS/WapPush sending	●	
Advertising actions/loteries	●	

Kaina	SMS	WAP	Prenumeratos
0.30 EUR	●	●	●
0.40 EUR	●	●	●
0.50 EUR	●	●	●
1.00 EUR	●	●	●
1.20 EUR	●	●	●
1.50 EUR	●	●	●
2.00 EUR	●	●	●
3.00 EUR	●	●	●
3.20 EUR	●	●	●

Schema 3 Supported prices in Estonia

Poland

Service	Supported	Comments
Short number charging	●	Short numbers: 70068, 71068, 72068, 73068, 74068, 75068, 76068, 79068
Periodical charging (subscriptions)	●	Just ERA users (WAP subscriptions)
WAP charging	●	Just ERA users
WAP identification	●	Just ERA and Tele2 users
SMS/WapPush sending	●	
Advertising actions/loteries	●	

Price	SMS	WAP	Subscription
0.61 PLN	●	●	●
1.22 PLN	●	●	●
2.44 PLN	●	●	●
3.66 PLN	●	●	●
4.88 PLN	●	●	●
6.10 PLN	●	●	●
7.32 PLN	●	●	●
10.98 PLN	●	●	●

Schema 4 Supported prices in Poland

Indonesia

Service	Supported	Comments
Short number charging	●	
Periodical charging (subscriptions)	●	WAP users must enter their MSISDN and password at project side. Password is generated and sent to user when registering to subscription via short number.
WAP charging	●	
WAP identification	●	MSISDN is not identified
SMS/WapPush sending	●	
Advertising actions/loteries	●	

Price	SMS	WAP	Subscription
500 IDR	●	●	●
1000 IDR	●	●	●

1300 IDR	●	●	●
2000 IDR	●	●	●
3000 IDR	●	●	●
5000 IDR	●	●	●
8000 IDR	●	●	●
10000 IDR	●	●	●
15000 IDR	●	●	●
20000 IDR	●	●	●

Schema 5 Supported prices in Indonesia

FAQ (frequently asked questions)

How to install the SMS subscription?

- Order the subscription service (for renewals) and SMS keyword services (for subscription and unsubscription) or complete the order form "Uzsakyti_SMS_prenumerata.xls"
- Get familiar with the Chapter Subscription Billing Services (SMS)
- Install (develop) a script located at URL that handles subscription reports and processes user subscriptions/unsubscriptions/renewals, etc.
- (Optional) Your service may require to install (develop) a script located at URL that processes SMS subscriptions and verifies whether the user is allowed to subscribe after the subscription SMS is sent.

How to install WAP subscription?

- Order the subscription service (for renewals), the WAP payment service (for subscription and unsubscription) and the WAP identification service (to identify the browsing user and its status), or complete the order form "Uzsakyti_WAP_prenumerata.xls";
- Get familiar with the Chapter Subscription Billing Services (WAP);
- Create (develop) a page with subscription rules, subscription and unsubscription;
- (Optional) Your service system may require a WAP user identification, in the course of which you receive not only information about the user (country, operator, etc.) but also the latest state of the subscriber. As a result, the subscription button will not be showed to active members.

What file is required for the billing operation and where should it be located?

Unfortunately, the installation of the billing system requires development skills and library classes need to be customised to the operational logic of your project/system. Short and simple examples are presented in the PHP programming language in the directory *MBSLib/examples/simple*. If necessary (if you do not have your developers), we may help you to install the library for your project. To agree on the number of hours and hourly rate, please e-mail us at support@vero.lt.

What is the purpose of the MBS PHP library?

This library helps the partner's developers to integrate the MBS system in their system:

If you develop in the PHP language (in both PHP4 and PHP5 environments);

If you do not want to use our library, you should implement those exemplary functions that are necessary in your case;

If you develop by any other technology, you can use the library or its separate functions as a technical

example.

Why WAP subscription needs two services (WAP and subscription)?

A subscription service is used for regular payments (renewals) only, however, in the case of WAP projects, users must subscribe to a service. Here, a WAP service, which helps the user to subscribe, regulates additional subscription rates, etc., comes into use.
Similarly with the SMS subscription. You create an SMS keyword service which subscribes a user to the service.

Why is it not possible to identify users when programming by Ruby language?

When using Ruby, REMOTE_ADDR may be found out with function remote_ip(). Quotation:
"The remote IP address. REMOTE_ADDR is the standard but will fail if the user is behind a proxy. HTTP_CLIENT_IP and/or HTTP_X_FORWARDED_FOR are set by proxies so check for these before falling back to REMOTE_ADDR. HTTP_X_FORWARDED_FOR may be a comma-delimited list in the case of multiple chained proxies; the first is the originating IP"

What is a channel for?

A channel helps to set different prices for one service (e.g. you can sell a service *Melodies* to omnitel_lt operator for LTL 1 or LTL 5). A channel is also useful because of its clear statistics. For instance, registered members (a channel for registered members) as well as non-registered members (a channel for non-registered members) can by your melodies. The statistics may also show how much you earn from a service and how much you earn from registered/non-registered members.
Why a channel is useful? Let's say you want to sell a service *Mp3 Melodies*. Melodies may be downloaded by registered and non-registered members. A channel 'wap.page.lt members' is for registered members and a channel 'wap.page.lt' is for non-registered members. Advantages:

- You can find out in which part of the site the user bought the service.
- The statistics show whether it is more beneficial to sell the service to registered or non-registered members.
- **You can have different prices for one service with different channels!** (e.g. EUR 1 for registered members and EUR 2 for non-registered members).
- The channels created may be used for other services.

WEB Billing Services (SMS Bank)

Pending to translate… (documentation is ready in Lithuanian)

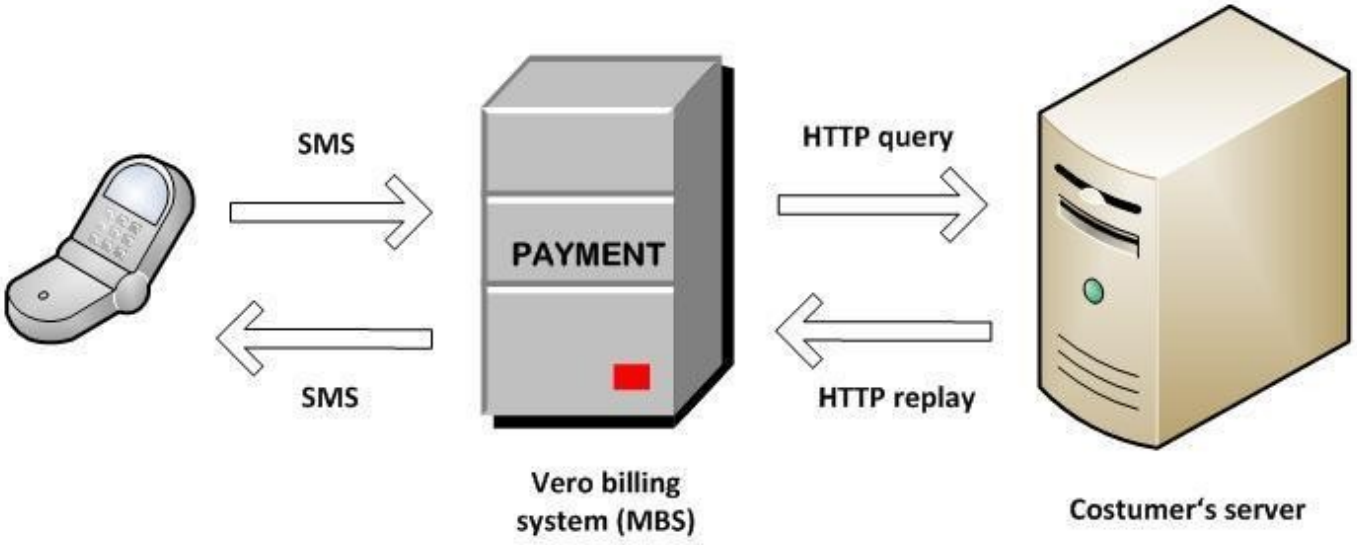
Keyword services

A keyword service is used for charging users by SMS messages. The user sends an SMS message, the content of which is processed in the MBS server. The service owner is informed about the SMS message received by the means described in Chapter 'Query Format'. The service owner sends to the Service Provider a message to be received by the user.

A simple example: a charity project. The user sends an SMS message with a keyword 'SUPPORT' (with no commas, small caps are also possible). The user is charged EUR X and receives a reply message with text 'We appreciate your support'.

To make everything easier, a package of the most frequent keyword services is made and grouped by operation.

Data transmission to the Partner



Schema 6Billing by short telephone numbers, and data transmission to the Partner

At first, the user sends an SMS message which is received by the MBS server. MBS charges for the message. A keyword helps to identify the service owner, which is informed about the message received by HTTP GET request. The service owner processes the message and replies to MBS by HTTP response, which may determine the reply message text.

Query format

MBS informs the Partner's server about the receipt of the SMS message by HTTP request at the address agreed on during the service request. The data is provided in GET data array:

Title	Description	Example
From	Service Provider's description	'vero'
action	Parameter showing the incoming message request	'sms'
ModuleName	Keyword identifying the service	'keyword1'
Msisdn	User's telephone number	37068714589
Phone	User's (SMS sender's) mobile phone number. The last 8 digits are displayed.	68714589
Number	Short telephone number whereto the message was sent (4 digits)	1679
Operator	Operator's name consisting of the operator's name, '_' and country abbreviation.	'omnitel_lt'
Provider	User's operator	'omnitel'
Country	Country code	'lt'
Sms	Full message text (max.160 symbols).	
msgId	Message identifier which is unique for every incoming SMS message	1234567890
retry	Number of the SMS message sending retries	0
price	Price in cents	50
currency	Price currency (established by ISO 4217)	LTL
s1 s2	Security signature (lower security level) Security signature (highest security level)	

Example of the created address whereto SMS data will be sent (the security signature uses the password 'test'):

```
http://mbs.vero.lt/simple/notify_sms.php?
From=vero&action=sms&ModuleName=test&Msisdn=37069923232&Phone=69923232&Number=1679&Oper
ator=bite_lt&Provider=bite&Country=lt&Sms=TEST+tekstas&TransId=fdda7e2ef38ea584e36ffc5c
a34d77013bb0e062&msgId=13861100&smsc=bite&s1=c7de429de90bc706a36d874284bcc1acf1bf6961&s
2=zxgX6hi7s5UKN5zKAihxcpm7PMnhQho2OEdY58T7jnsts52jDUNA
%2FoXltbCWd0BoIt1uMee7Nmkn8dQpwQJq96PbKFaQ04DaU
%2B9xVjVXySeavlhL8%2FdOCzo0lqpPLskAN5YijWfxmbzizRWTgJnO1KdEWz49vtpAaQwFyTyXYG4%3D
```

Following the HTTP GET query, the Partner must create a reply in a text line with the format '{RESULT};{PARAMETER1};{PARAMETER2};{PARAMETER3}'. Logical parts of the reply message are separated by ';', and all parameters are encoded (urlencode). A list of possible reply messages:

RESULT	Description	PARAMETERS
No reply	1. The Partner **did not accept the**	-

	request. 2. The **service** provision is **unsuccessful**.	
ERROR	1. The Partner **accepted the request**. 2. The **service** provision is **successful**. 3. A reply message informing about the error is sent to the user.	1. *sms* – Reply message text about the error
SMS	2. The Partner **accepted the request**. 3. The **service** provision is **successful**. 4. A **reply SMS message** is sent to the user.	1. *sms* – reply message text; 2. *ident* – (not mandatory) if the Partner wants to have a message identifier, e.g. advertisement ID
PUSH	1. The Partner **accepted the request**. 2. The **service** provision is **successful**. 3. A **reply WapPush message** is sent to the user.	1. *url* – URL of the reply message; 2. *sms* – reply message text; 3. *ident* – (not mandatory) if the Partner wants to have message identifier, e.g. advertisement ID
NONE	1. The Partner **accepted the request**. 2. The **service** provision is **successful**. 3. No **reply SMS message** is sent to the user.	1. *ident* – (not mandatory) if the Partner wants to have message identifier, e.g. advertisement ID
BILL	1. The Partner **accepted the request** and wants to control the billing price. 2. The **billing result** will be delivered together with the second HTTP request. 3. The service is **provided after** the billing result.	1. *price* – billing price in cents; 2. *currency* – billing currency; 3. *ident* – (not mandatory) if the Partner wants to have message identifier, e.g. advertisement ID

If the reply is not created, MBS believes the Partner's script is not operating and sends to the user a reply message informing that the Partner's service is unavailable: 'Currently the server is not available, please try later'.

Example

You can try the operation of the service created (but not fully connected) by sending the keyword 'TST' at a short number 1676 (message price LTL 0.30). You type the keyword, leave a space and type HTTP address whereto the message will be redirected. If the address does not begin with 'http://', it is automatically added.

Thus, for instance, if you send a message 'Tst mbs.vero.lt/simple/norify_sms.php' at 1676, you receive a message '"s" is missing at the end of the message'.

Such text is returned as the following file is placed at http://mbs.vero.lt/simple/norify_sms.php:

If you do not want to use the MBS PHP library, you should implement your own *verifyArrayHigh* function.

```
<?php
/**
 * Raw SMS keyword notify message receiving example
 *
 * @see MBSLib/examples/simple/notify_sms.php
 *
 * @package RawExamples
 * @author Liutauras Ričkus <lr@vero.lt>
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// MBS function library is initiated
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');
$mbs_base = new MBSBase();
$mbs_base->setSecretKey('test');

// Configuration
$config['servers'] = array('213.226.139.33', '213.226.139.42', '213.226.139.43');

// The main SMS data is read
$from = $_GET['From'];
$module = $_GET['ModuleName'];
$msisdn = $_GET['Msisdn'];
$number = $_GET['Number'];
$operator = $_GET['Operator'];
$sms = $_GET['Sms'];

// GET parameters of unknown server
if ( !in_array( $_SERVER['REMOTE_ADDR'], $config['servers'] ) ) {
          echo 'Klaida: nezinomas serveris';
          exit();
}

// Falsified GET parameter
if( !$mbs_base->verifyArrayHigh($_GET) ) {
    echo 'Klaida: siuntejas ne MBS';
    exit();
}

// Service logic
if ( substr($sms, -1)=='s' ) {
          $response = array('SMS', 'Letter "s" found at the end of message');
} else {
          $response = array('SMS', 'No letter "s" found at the end of message');
}

// Service reply to the user
echo implode(';', array_map('urlencode', $response));
```

```
?>
```

Billing management

First the user is charged for the service (e.g. advertisement) and only then it is provided. For example, after the billing process, it turns out that the user's message contained some mistakes, e.g. incorrect advertisement number. In order not to charge such users, it is necessary to create a free keyword service and charge users only after the message is verified. In such a case, the Partner uses BILL-type reply with a payment price included (if the user makes a mistake, ERROR-type reply is used).

Following the BILL-type reply, the user is charged by MBS the price indicated. Only after the successful charge, MBS sends the second HTTP request to the Partner's server. The data of this request is the same as of the previous one, only some additional data are added:

Title	Description	Example
action	Parameter showing the billing result request	delivery
ident	Message identifier provided by the Partner, e.g. advertisement ID	12345
status	The key status of the transaction, showing whether the transaction was successful. A list of possible meanings is presented in Chapter 'Transaction Statuses and States'	commit
state	The additional state of the transaction, specifying the reasons for the unsuccessful transaction. A list of possible meanings is presented in Chapter 'Transaction Statuses and States'	op_done

The Partner can decide when and how to charge the user. In such a case, the service is provided only after a successful (*commit*) billing result.

Example

For example, in the case of a game credit purchase, the user buys as many credits as specified in the message:

- The user sends a message with text '`TEST1 50`';
- The user is charged a minimum price, e.g. LTL 0;

Message data is transferred to the game website (the security signature uses the password 'test"):

```
http://mbs.vero.lt/simple/notify_sms2.php?
```

```
From=vero&action=sms&ModuleName=test1&Msisdn=37069923232&Phone=69923232&Number=1679&Ope
rator=bite_lt&Provider=bite&Country=lt&Sms=TEST1+50&TransId=fdda7e2ef38ea584e36ffc5ca34
d77013bb0e062&msgId=13861109&smsc=bite&s1=a6b0f603ef7850ec9a95d9a34e03b207d3c59f5d&s2=W
JzvpB40NG9ga8DgE6dPkgtEmFP3o1dUI8TivLnITgqp%2FobqWAaH7EcQ%2BsZj
%2FdkR51N6udp2SVZdVdyfuNT5zmEvJ6mMufwU5%2B7%2Fd96gYfz56EHuyuvZtfPJ%2F1e%2B%2BgyvI5Z
%2B7IEevxP5SE5o7MW4zBXF9bKq0HfpFD%2FR1Snf%2F48%3D
```

The game website checks if the user entered the allowed number of credits and replies '`BILL;50;LTL`' (in case of error '`ERROR;Not+specified+how+much+credits+to+buy`');

The user is charged the amount set, i.e. LTL 0.50;

The billing result is transferred to the game website (the security signature uses the password 'test'):

```
http://mbs.vero.lt//simple/notify_sms2.php?
From=vero&action=delivery&ModuleName=test1&Msisdn=37069923232&Phone=69923232&Number=167
9&Operator=bite_lt&Provider=bite&Country=lt&Sms=TEST1+50&TransId=fdda7e2ef38ea584e36ffc
5ca34d77013bb0e062&msgId=13861109&smsc=bite&ident=&status=commit&state=op_done&s1=9b205
bfb6836921b28bab62dff6b4dc9a3b8ada1&s2=UqtwsrnsjZDmTnqnvbwTWFErEma8b2JD85EMV0xv9gCHn
%2BiAnMf%2FGyypCMezeUhqctlP8qADnsYEF%2BzGq6OfP
%2FWx8BYAtj0kLuHNyOh4Ggim0%2BsAUFKxsUbqFp2%2F1ndCHrTlNi
%2FZmi4qliCrvVlImrv41VUQheNeVSGR3TP%2BARc%3D
```

The game website activates the credits purchased and creates a reply message to the user: '`SMS;You+purchased+50+credits`'.

The example is realized using MBS PHP library:

If you do not want to use the MBS PHP library, you should implement your own *verifyArrayHigh* function.

```
<?php
/**
 * Raw SMS keyword notify message receiving example (user price depends on dynamic
condition)
 *
 * @see MBSLib/examples/simple/notify_sms2.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */
```

```
//** ... the code from the previous example must be here ... **/

// Service logic
list($keyword, $credits) = explode(' ', $sms);
switch ($action) {
    // sms received
           case 'sms':
        // Charge by dynamic condition
                        if( is_numeric($credits) ) {
            $response = array('BILL', $credits, 'LTL');
        // SMS text is out of order
                        } else {
           $response = array('ERROR', 'Not specified how much credits to buy ');
        }
    break;

    // Billing result received
           case 'delivery':
        /** ... the service is provided here ... **/
        $paslauga_suteikta = true;

                        $response = array('SMS', 'You purchased '.$credits.' credits');
    break;
}

// Service reply to the user
echo implode(';', array_map('urlencode', $response));

?>
```

Data transmission to the Partner (with retries)

Such service is used, if the Partner's server is unavailable for a short period of time. The principle of operation is the same one used in the case of simple data transmission, only the data transmission

is repeated the number of times set or until the Partner's server successfully receives the message. For example:

- The user sends an SMS message with keyword the 'TST'. The data is transmitted to the Partner's server;
- During the data transmission, the Partner's server is unreachable because of HTTP request timeout. In such a case, a reply message is not sent to the user;
- The data transmission regarding the incoming SMS message is repeated every minute until it is successful;
- If the data transmission is not successful 5 times in a row, the data transmission is cancelled and a reply message is sent to the user, e.g. 'The server is unreachable. For service provision please contact …'. A message about the error may be also sent to the Partner's e-mail.

The repetition of data transmission may result in providing the service more than once. For example, the user's account is toped up twice even though only 1 message was sent. This may happen when the Partner's server processes the data received but fails to send a reply on the receipt of the request in time (HTTP request timeout). As a result, it is necessary to verify that the re-sent message has been already processed:

- The parameter msgId is unique for every incoming SMS message;
- The parameter 'retry' shows the number of times the data transmission is performed.

Questions – answers

The following games are often developed:

- A game question is advertised: 'Who likes honey? A – Bees, B – Winnie-the-Pooh, C – Bucket';
- The user sends a message with text 'QUESTION answer letter'. A reply message is sent to the user informing if his/her answer was correct;
- The customer may review game participants, select the ones with the right answer and choose the winner.

Such service is also used in cases where the Partner does not require integration on its side. Only the incorrect answer with no options is given (e.g. donation). The users send a massage and immediately receive a reply message. The Partner may review the messages received in reports.

Subscription Billing Services (SMS)

Subscription billing is used for automatic billing for club-type services. A fixed fee is automatically charged from the service subscriber's account on a regular basis.


In case of all subscriptions, the service provider (partner) does not have to calculate paid or renewed period because that is what MBS is for. If the member is active in the MBS subscription service, he/she is active in the partner's website as well (and vice versa).

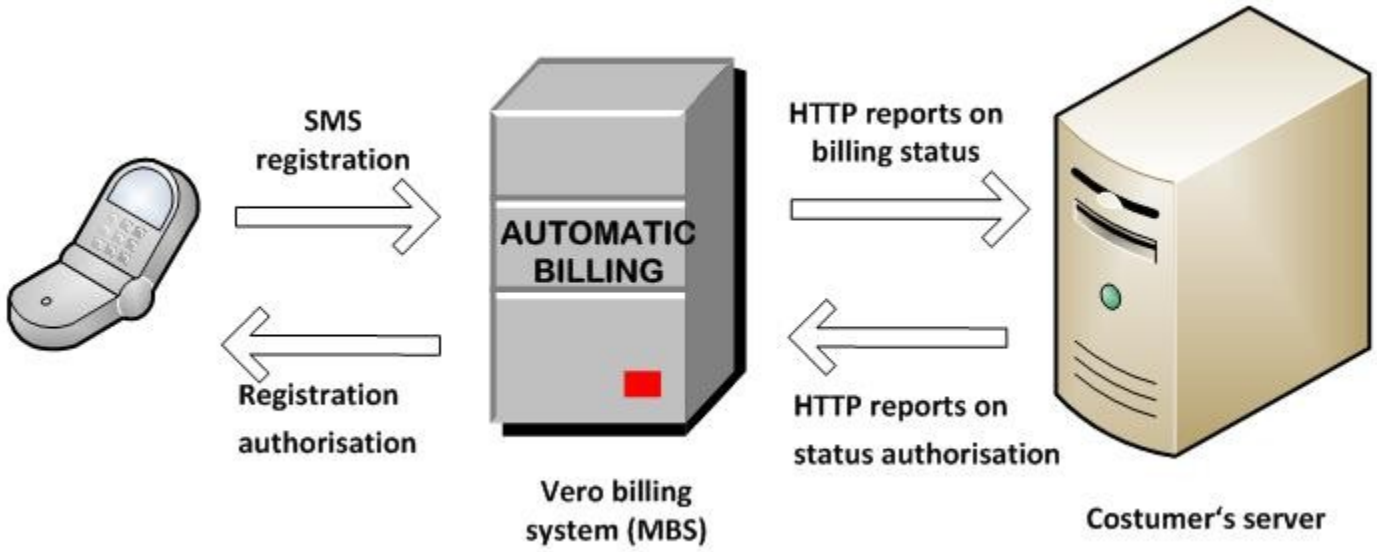

In Lithuania (Tele2, Omnitel), users are charged for subscription services only from 10.00 to 20.00. Users the subscription of which end between 20.00 and 10.00 are charged next morning from 10.00.


In case of operator "Omnitel Lietuva", users are removed from subscription if they did not used service for a month. External system provides user's last project visit date in response of "check" report (see Reports to the external system).
If services do not provide user's visit date, users of such services must approve each renewal by sending SMS message YES

Let's look at one of the most popular examples. The membership costs LTL 1 per week. The user subscribes to the service and the membership fee is automatically charged every week. Subscription services are classified by the subscription method:

- **SMS subscription** – users subscribe and unsubscribe by sending an SMS message at short number (for more see this Chapter).
- **WAP subscription** – users subscribe and unsubscribe by browsing a WAP site (for more see the next Chapter).

SMS subscription is used in cases where the user subscribes to a service after a keyword sent by him/her is received. If the user has already subscribed to the service, his/her subscription is renewed for another period (e.g. one week).



Schema 7SMS subscription and billing

The figure illustrates SMS subscription and billing:

- At first, the user sends an SMS message which is received by the MBS;
- The user is informed on the successful or unsuccessful subscription by an SMS message;
- The user is charged on a regular basis set by the service parameters.

Users of Bitė Lietuva and Bitė Latvija are subject to different rates because of a monthly subscription period. The price and the period must be indicated separately.

Reports to the external system

The partner's service is notified by HTTP queries of the new user, unsubscription or suspension of the membership. The user may perform the following actions in the subscription:

Action	Description	PARAMETERS
sms	Query before the SMS subscription (see Order of SMS subscriptions)	Depends on service configuration: 1. *sms* – Possibility to control reply SMS message or fragment of it; 1. *sdata* – Possibility to control external system identifier which is used to order subscription;
register	New user has subscribed to the service.	-
suspend	The user's membership is temporarily suspended.	-

resume	The user's membership is resumed after temporary suspension (billing).	-
pay	The user's subscription is renewed (billing).	-
remove	The user has unsubscribed from the service.	-
approve	User approved subscription registration by message YES - for "Omnitel Lietuva" users only (turned on by partner request)	-
approve_renew	The user approved subscription renewal by message YES - for "Omnitel Lietuva" users only (turned on by partner request)	-
check	Checking when this membership was used in then project-side the last time - for "Omnitel Lietuva" users only (WEB visit, login to the account etc.)	After receiving request "check" external system must provide user's data in project-side: 1. *last_login_date* – date of last visit in project. Or empty string "" if unknown; 2. *last_login_ip* – IP from which user logged in. Or empty string "" if unknown; Example: `OK;2012-02-10+17%3A18%3A05;81.7.76.85`

Figure 8 User's actions in subscription

Data are presented by the HTTP GET method:

Title	Description	Example
action	Name of the action	resume
serviceID	Unique service ID	11
mbs_account_id	Unique user's identifier in the MBS	1
mbs_account_phone	User's telephone number (MSISDN)	37069923232
mbs_account_ident	WAP user's identifier in the operator's system (unique in each operator's system). It may be a telephone number, but only in rare cases.	2219983
operator	Name of the user's operator consisting of the operator's name, '_' and a country code.	bite_lt
provider	Name of the user's operator in the country	bite
country	Country code	lt
sdata	(Optional) User's identifier in the partner's system	up to 50 symbols
id	Unique query ID	15
dateAdd	Date of the action by minutes	200604251008
status	(Optional) Code of the error that resulted in the suspension of the membership. The parameter appears only where action=suspend.	98 – The user's account is empty or the error occurred in the operator's

		billing system; 99 – The user has exceeded the monthly account limit.
next_bill	Preliminary date of the next renewal (format YYYY-mm-dd HH:ii:ss). The date is preliminary as subscriptions are not renewed at nighttimes.	2009-12-28 12:28:30
price	Price in cents shows how much user was charged (given for reports 'register', 'pay', 'resume')	100
currency	Currency is sent with parameter 'price'	LTL
s1	Security signature (lowest security level)	-
s2	Security signature (highest security level)	-

Example of the HTTP query:

```
http://mbs.vero.lt/simple/notify_subscription.php?
action=register&serviceID=64053&mbs_account_id=940195&mbs_account_phone=37065902420&mbs
_account_ident=&operator=omnitel_lt&provider=omnitel&country=lt&memberID=32457&msisdn=3
7065902420&phone=65902420&dateAdd=201007151016&price=50&currency=LTL&key=nDrSNh2wq60%3D
&id=17473575&sdata=fkx9&s1=15d080e4aa13c92ccf460db50a27fa670e49fc73&s2=yGEV0Y59%2Fx4xHn
mSbXZ7qUErXaAYz9locCwiDRzbGxUZMCTmb0MIDjXpMrK4ngrKpdeOdr5yXTIZoeckYJUJjeuUXle5UjqZQ8v4o
z9M3d%2F0aIMJxuzR521xC%2BLkK%2BphFyVLrKW2Kv0arMdQyOeQsH7hdWTyoF%2FGaCLxZ66%2F280%3D
```

Following the HTTP GET query, the Partner must create a reply in a text line with the format '{RESULT};{PARAMETER1};{PARAMETER2};{PARAMETER3}' (without quotation marks). Logical parts of the reply message are separated by ';', and all parameters are encoded (urlencode). A list of possible reply messages:

RESULT	Description
OK	1. The Partner **accepted the request**. 2. The **service** provision is **successful**. 3. The request **will not be repeated**. Most of actions don't need parameters and it is enough to reply by "OK" (without quotation marks). If parameters are needed: `„OK;2012-02-10+17%3A18%3A05;81.7.76.85“`.
ERROR=NOT_MEMBER	1. The Partner **accepted the request**. 2. The **service** provision is **unsuccessful**. 3. The request **will not be repeated**. If the partner's server receives a query about a user who is not in the partner's system (e.g. incorrect user's identifier in the partner's system), the text "ERROR=NOT_MEMBER" (without quotation marks) should be output and the user automatically removed from subscription so that he/she would not pay for the service he/she does not receive.

ERROR	1. The Partner **did not accept the request**. 2. The **service** provision is **unsuccessful**. 3. The request rejected and **will be repeated**. If the error occurs in the partner's server, it is recommended to output an information error message, e.g. "ERRORFailed connection".
Unexpected reply	1. The Partner **did not accept the request**. 2. The **service** provision is **unsuccessful**. 3. The request rejected and **will be repeated**. If the reply OK is not received, it is considered that the connection to the server failed and the same query is repeated within 3 minutes.

Example of the report processing

The example is implemented using the MBS PHP library:

If you do not want to use the MBS PHP library, you should implement the function *verifyArrayHigh()*.

```
<?php
/**
 * Raw subscribtion notify messages receiving example
 *
 * @see MBSLib/examples/simple/notify_subscription.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// Inicijuojame MBS funkcijų biblioteką
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');
$mbs_base = new MBSBase();
$mbs_base->setSecretKey('SuperSecure');

/**
 * Suranda sistemos naudotoja pagal nika
 *
 * @param string $nick Sistemos naudotojo nikas
 *
 * @return array Naudotojo duomenu struktura arba NULL jei vartotojas nerastas
 */
function find_user($nick)
{
    if (in_array($nick, array('a', 'b', 'c'))) {
        return array(
            'nick' => $nick,
            'state' => 'inactive',
            'credits' => 0,
            'wait_for_pay' => false,
            'last_login' => '2012-02-10 17:18:05',
            'last_login_ip' => '81.7.76.85',
        );
```

```
    } else {
        return null;
    }
}

// Konfigūracija
$config['servers'] = array('213.226.139.33', '213.226.139.42', '213.226.139.43',
'193.105.49.4', '193.105.49.5');

if ( in_array( $_SERVER['REMOTE_ADDR'], $config['servers'] ) ) {
    if ( $mbs_base->verifyArrayHigh($_GET) ) {
        // Nusiskaitome pagrindinius pranesimo duomenis
        $action = $_GET['action'];
        $mbs_service_id = $_GET['serviceID'];
        $mbs_account_id = $_GET['mbs_account_id'];
        $mbs_account_phone = $_GET['mbs_account_phone'];
        $nick = trim($_GET['sdata']);

        // Uzklausimas, pries vartotojo SMS registracija
        if ($action=='sms') {
            /** ... cia tikriname ar zinutes siuntejui leisti registruotis ... **/
            if (strlen($nick)>0 && find_user($nick)) {
                $response = 'OK';
            // Formuojame atgalines zinutes atsakyma
            } else {
                $response = 'ERRORSuklydote irasydamas savo nika, pasitaisykite';
            }
        // Vartotojo veiksmai prenumeratoje
        } else {
            // Surandame vartotoja savo sistemoje
            if (strlen($nick)>0 && $user=find_user($nick)) {
                switch ($action) {
                    // Uzsiregistravo naujas vartotojas
                    case 'register':
                    // Vartotojo naryste atnaujinta po laikino sustabdymo
(apmokestinimas)
                    case 'resume':
                        /** ... cia vartotojui suteikiama VIP prieiga ... **/
                        $user['state'] = 'active';
                        $user['credits'] += 10;

                        $response = 'OK';
                    break;

                    // Vartotojo prenumerata pratesta (apmokestinimas)
                    case 'pay':
                    case 'renew':
                        /** ... cia renkama statistika, kiek ivyko sekmingu pratesimu
(VIP prieiga jau turetu butu suteikta) ... **/
                        $user['state'] = 'active';
                        $user['credits'] += 10;

                        $response = 'OK';
                    break;
```

```
                    // Vartotojas issiregistravo
                    case 'unregister':
                    case 'remove':
                    // Vartotojo naryste buvo laikinai sustabdyta
                    case 'suspend':
                        /** ... cia vartotojui sustabdoma VIP prieiga ... **/
                        $user['state'] = 'inactive';

                        $response = 'OK';
                    break;

                    // Tik Omnitel, patvirtinus prenumerata (ijungiama partnerio
prasymu)
                    case 'approve':
                    case 'approve_renew':
                        $user['state'] = 'active';
                        $user['wait_for_pay'] = true;

                        $response = 'OK';
                    break;

                    // Tik Omnitel, tikrinama kada paskutini karta si naryste naudota
projekto puseje
                    case 'check':
                        /** ... cia grazinama paskutine vartotojo prisijungimo data ir
IP ... **/
                        $response = array('OK', $user['last_login'],
$user['last_login_ip']);
                    break;

                    // Nenumatytas pranesimo tipas
                    default:
                        $response = 'ERRORNezinomas pranesimo tipas';
                    break;
                }
            // Jei gaunama uzklausa apie vartotoja, kurio nera partnerio sistemoje
(pvz. klaidingas vartotojo identifikatorius partnerio sistemoje)
            } else {
                $response = 'ERROR=NOT_MEMBER';
            }
        }
    // Sufalsifikuoti GET parametrai
    } else {
        $response = 'ERRORSiuntejas ne MBS';
    }
// Nežinomo serverio GET parametrai
} else {
    $response = 'ERRORNezinomas serveris';
}

// Paslaugos atsakymas vartotojui
if (is_array($response)) {
    echo implode(';', array_map('urlencode', $response));
```

```
} else {
    echo $response;
}

?>
```

Order of SMS subscriptions SMS

Usually the user is subscribed to the service immediately after the subscription message is sent. The partner is notified thereof by a report on the ***register*** action.

Sometimes the partner may have to dynamically form a reply SMS message or its part (e.g. login password) or simply to check if the user may be registered (e.g. if the username indicated in the SMS message does not exist in the partner's system). For this purpose, a report on the ***sms*** action is sent. The request is made using the HTTP protocol (the same as in the Reports to the External System) with several additional data included:

Title	Description	Example
From	Description of the service provider	vero
ModuleName	Keyword to identify the service	keyword1
Number	Short telephone number to which the message is sent (4 digits)	1679
Sms	Full message text (no longer than 160 symbols)	Pazink 12345 Mano vardas auksinis kardas
msgId	Message identifier which is unique to every incoming SMS message	1234567890

For example, the user wants to get the Golden Membership for a single or several profiles on the community site. He/she sends an SMS message with the text "CLUB 'Username'". The MBS sends a query to the partner's server to find out if the SMS sender can be registered. The partner's server receiving the subscription SMS data may perform the following actions:

- to check if the username indicated by the user already exists in the system;
- to generate a dynamic reply SMS text or any of its parts, e.g. login password;
- to inform the MBS if the user's registration should be continued;
- to save data of the attempted registration.

Let's continue with the example about a community site:

- If the server replies “OK, the password of the golden login is ADR123”, the user is subscribed to the service with the same “sdata” variable that is indicated in the SMS text. The user receives a reply SMS message with the text indicated by the server;
- If the server replies “ERROR incorrect username. Please indicate the correct one”, the registration is terminated. The user receives a reply SMS message with the text indicated by the server.


Sometimes the customer’s server sends not the reply SMS text but the generated “sdata” parameter with which the user is registered in the subscription (e.g. the username on the community site is found with a help of the telephone number, etc.).


Sometimes it is necessary to terminate certain subscriptions before registration. Thus, for instance, if the user orders the Platinum service level, Golden or Silver levels are terminated (if the user has such memberships).

Refusal of SMS subscriptions

The service is used in cases where a single or several subscriptions are terminated after a keyword from the user is received. The partner is notified thereof by a report on the ***remove*** or ***unregister*** action.

Examples of SMS subscriptions

User identification in the partner’s system by telephone number

Let’s say, a community site, the membership of which costs LTL 5 per 48 hours, allows using its services without limitations. Two VERO systems are used for the implementation of such service:

1. The regular billing (subscription) system charges LTL 5 from the telephone numbers registered in the subscription every 48 hours. At every stage of its operation the system may inform the community site about its actions with every subscriber (reference http://www.partneris.lt/?getSMS?). For example:
   On 2006-05-33 14:18 tel. No +37068261653 successfully subscribed to the service id=14: http://www.partneris.lt/?getSMS?action=register&serviceID=14&memberID=4252&phone=65293683&dateAdd=200605221418&key=xesluQ%3D%3D&id=1260&provider=bite.
   The next day the same telephone number unsubscribes from the service. LTL 5 is not charged for that

day, however, the user pays LTL 5 for the subscription SMS.

http://www.partneris.lt/?getSMS?action=remove&serviceID=14&memberID=4252&phone=65293683&dateAdd=200605231121&key=xesluQ%3D%3D&id=1349&provider=bite

The community site application having received a ***register*** query allows the owner of this telephone number to use its services for the unlimited time. After one day the application receives information that the user has unsubscribed from the service (***remove*** query) and terminates the service provision. Similarly, the membership should be terminated after receipt of the ***suspend*** query (LTL 5 could be charged for the upcoming 48 hours) and resumed after receipt of ***resume*** query (when the said user is finally charged). The community site considers that VERO regular billing system charges and provides its services to all telephone numbers, the membership of which is not suspended.

2. How telephone numbers end up in this system? All the work is done by another VERO system, i.e. the SMS service system. When the user sends an SMS message with a keyword ACQUAINTANCE, the following steps are made:
   - The system checks if the telephone number already belongs to the existing service (if it does, the membership is renewed for another payment period as the user pays LTL 5 for this SMS message either way);
   - If the telephone number does not belong to the existing service, a reply service “Thank you, you have successfully subscribed to the service” is sent (such user is then charged after 48 hours).

   The MBS and the system of the community site may group everything into the following steps:
3. The user sends an SMS message with the text ACQUAINTANCE.
4. The SMS service system processes the SMS message and includes the telephone number into the appropriate service in the Regular Billing System.
5. If no errors have occurred, the application http://www.partneris.lt/?getSMS starts receiving messages of the Regular Billing Services about the registration of the telephone number (***register*** message). If a reply OK is not received, messages are repeated until the partner’s application finally receives them.
6. After 48 hours the Regular Billing System will try to charge this telephone number. If the billing is successful, http://www.partneris.lt/?getSMS receives a ***pay*** message; otherwise it receives a ***suspend*** message. When the system finally manages to charge the user after receipt of the ***suspend*** message, the partner receives a ***resume*** message. In this way it is possible to track every step of the user and make respective moves in accordance with the system scheme (mostly, it is resume–suspend the provision of a

certain service).

7. The user decides to unsubscribe from the service and sends an SMS message with the text NO ACQUAINTANCE. VERO SMS Service System checks if the sender's telephone number belongs to the service provided by the Regular Payment System and tries to unsubscribe it from such service.
8. If the unsubscription process is unsuccessful, the Regular Billing System starts sending ***remove*** messages to http://www.partneris.lt/?getSMS and waits for the reply OK.

User identification in the partner's system by the partner's identifier

This scenario is suitable for connections where the user is identified in the service by the separate identification system created by the partner rather than by the telephone number. In this way one telephone number may subscribe to the same service several times. An ad site may serve as a perfect example. One user can have N advertisements and pay for them by sending SMS messages with the ad ID indicated. In this case the ad ID is the identifier in the partner's system.

Technically, this may be implemented using two systems – the simple keyword management system and the subscription system. Step-by-step subscription to/unsubscription from the subscription ad system:

1. The user sends an SMS message.
2. The keyword system sends the message to the partner's server (see SMS Service System) and receives from it the user's identifier (e.g. the ad ID) and, if required, a reply text.
3. The keyword system registers the user in the subscription system.
4. The subscription system starts sending standard-format messages to the partner's server. The HTTP GET parameter sdata is identified in the partner's system (e.g. the ad ID) and the partner's system knows the advertisement to which the action should be performed.
5. To unsubscribe from the system the user sends an SMS message with a respective keyword and the ad ID.
6. The keyword system addresses the subscription system and unregisters the user.

Subscription billing services (WAP)

The service is used for billing in the WAP environment when the user is not allowed to use WAP project services (or her/his possibilities are limited) unless she/he has subscribed to them. Once the user subscribes to the service, the MBS makes sure she/he receives it as long as the billing for the next period is successful. The regular fee is charged automatically as long as the user is subscribed to the service.

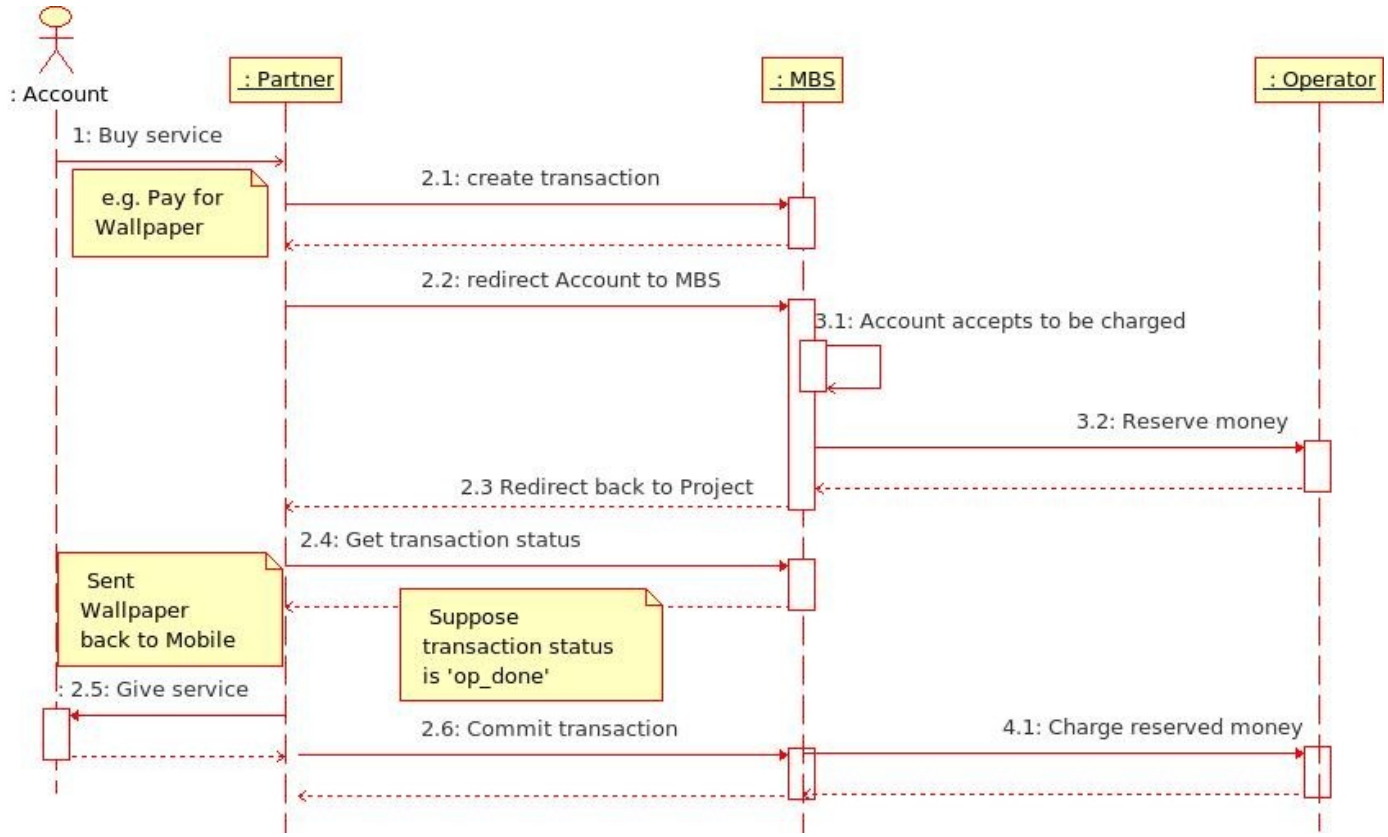

Before the subscription or right after the user's visit in the WAP project, we highly recommend using WAP identification service. Here you will find the detailed information about the user (country, language, operator, etc.) and you will be able to check the latest user's activity status in the subscription.


The user's subscription/unsubscrition is very similar to WAP billing service. Therefore, WAP service identifier is used for subscribtion.


User's renewals are monitored by the subscription service identifier.

Redirection to MBS for registration

The browsing user clicks on the link which redirects her/him to the MBS subscription environment. The user is asked if she/he agrees to subscribe to the service. If yes, the user is charged for the first period and is activated; if no, the subscription is terminated. Either way, the user is returned to the WAP site of the service. For more detailed information see the scheme below:



Schema 9 WAP subscription stages

- *The project* redirects *the user* to *MBS* with the subscription data (2)
- *MBS* work begins (3):
  - *(not mandatory) the user* confirms that she/he wants to subscribe to the service (3.1)
  - the user subscribes and the money is reserved for the first period
  - (not mandatory) *The project* is informed on the subscription
  - *MBS* returns the user to the project (3.3).
- *The project* makes sure the subscription has been successful (2.4) and provides the service to *the user* (2.5).

In case of the operator Omnitel Lietuva, during the billing procedure, the user is sent to the waiting page with a message '*The payment is in process. Once you receive the SMS message, click on this link: to continue*>'. A WapPush payment message is sent to the user, indicating the name of the service and a link to the partner's service. Hence, even if the user leaves the WAP browser, she/he will be able to continue the subscription by clicking on the link received by the WapPush message.

If partner receives payment errors, but error is not processed correctly (HTTP status is not 200), operator Omnitel Lietuva payments will fail!!!

To register the WAP user, she/he must be redirected to MBS at http://bill.vero.lt/wap/register with formatted GET array parameters:

Name	Type	Mandatory	Description
s	int	yes	WAP service ID for subscription
si	string	no	The user's identifier in the subscription service (e.g. password). It is used, if the same user subscribes to the service more than once but uses different identifiers.
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction). The parameter is also used to avoid double billing, therefore, each time it must be unique for one service or at least one user. If you do not store billing events, it is possible to show the time of the link generation.
p	int	no	Subscription (first period) price indicated in cents. It is requierd, if the subscription has several prices.
pc	string	no	Subscription currency (established by ISO 4217). It is required when specifying the price in parameter *p*.
u	string	no	Encoded (urlencode) URL, whereto the user is redirected after the subscription. In the absence of this parameter, the user will be redirected to the service URL indicated during the service creation.
un	string	no	Encoded (urlencode) URL, if it is necessary to redirect the user to a different URL after the unsuccessful subscription. In the absence of this parameter, the user will be redirected to URL indicated in parameter *u*.
s1	string	yes	Security signature (lower security level)

The example of a generated address (URL) whereto the user will be redirected (the security signature uses the password 'test'):

```
http://bill.vero.lt/wap/register?
session_id=debaf588e8ebadfe97a6ae38ff95beef&after_register=yes&s=2&si=&i=1255530969&u=h
ttp%3A%2F%2Fmbs.vero.lt%2Fsimple
%2Fwap_register.php&s1=01cb763307dd28851f8ccf3192b5fcbfd8ccd858
```

Additional parameters may be added. Later, together with the billing data they will be added to the address, using which the user with the subscription data will return to the project. Note: parameters cannot have such names as 's', 'si', 'i', 'p', 'pc', 'u', 'un', 's1', 's2', as these parameters are used for transferring the key information.

To avoid double billing, it is not enough to generate a billing link by clicking on a purchase button. In such a case, by clicking on the same button twice in a row, you will generate two different purchase requests and both of them will be paid for. We recommend making the purchase button a generated purchase link with the unique parameter *i*.

If you initiate WAP subscription via SOAP, detailed descriptions of how to initiate subscription events, verify subscription status, unsubscribe, etc. may be found in Annex 'Description of Remote Methods'

Redirection of the user to the external system after registration

Name	Type	Mandatory	Description
s	int	yes	WAP service ID for subscription
si	string	no	The user's identifier in the subscription service (e.g. password). It is used, if the same user subscribes to the service more than once but uses different identifiers.
sr	string	yes	The subscriber's key status which describes the activity of the user in the subscription. For a list of possible values see States and statuses of subscribers
srs	string	yes	The subscriber's additional state which specifies why the billing (subscription, renewal, etc.) is unsuccessful. For a list of possible values see States and statuses of subscribers
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction).
t	int	yes	Unique identifier of the billing event (transaction) in the MBS system
a	int	no	Unique identifier of the charged user in the MBS system (account_id)
s1	string	yes	Security signature (lower security level)
s2	string	yes	Security signature (highest security level)

The user is redirected to the service address after the (un)successful subscription (the link includes the parameters requested by the external system when initiating the event). Thus, for instance (the security signature uses the password 'test'):

```
http://mbs.vero.lt/simple/wap_register.php?
session_id=debaf588e8ebadfe97a6ae38ff95beef&after_register=yes&s=2&si=&sr=active&srs=ac
tive&i=1255530969&t=2331494&a=1&s1=dfbb0af21a09516b73946bb99458499f1ef6b081&s2=HpwXNO
%2Fr5Yhd3g00yIeHKEnZsuYJ7UdGTHnj06n5T%2FJQ
%2BDGZG8tPaMYH6rCI9Febv8UDtNgQLeXi7h21diQUxEirxNjb1AqdvSOi
%2FbO3IqCvuvGS65t7EIom5E9tGL77wdtorKpwgNkbfszQAgBEjard7scWiwt0Jbhut9ECDik%3D
```

When the user returns to the service address, it is necessary to verify the security signature (parameter 's1' or 's2'), thus ensuring that the user is redirected from MBS.

When the user returns to the service address, it is necessary to verify the subscriber's status (parameter 'r1' or 'srs') and make ensure that the service is not provided in case of the unsuccessful subscription.


When the user returns to the service address, you can check if the browsing user is the subscriber (parameter 'a'). This way you make sure that you do not provide the service even when the return of someone else's subscription is imitated.


Subscription report *register* is sent to the partner after a few seconds or minutes. However, some delays are possible.

Example of the WAP registration

The example (WAP subscription, the results of which are displayed on the screen) is implemented by using the MBS PHP library.


If you do not want to use the MBS PHP library, you should implement some functions for your convenience, e.g. *signArrayLow()*, *verifyArrayHigh()*, *MakeIdentify()* and *_getUserInformation()*.

```
<?php
/**
 * Raw WAP registration example (using redirect)
 *
 * @see MBSLib/examples/simple/wap_register.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

session_start();
include_once(dirname(__FILE__).'/../../lib/Identify.class.php');
include_once(dirname(__FILE__).'/../../lib/SBBS.class.php');

// MBS identification library is initiated
define('IDENTIFY_SERVICE_ID', 1);
define('IDENTIFY_BACK_URL', 'http://'.$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']);
define('IDENTIFY_SECRET_KEY', 'test');
$identify = new Identify(IDENTIFY_SERVICE_ID, IDENTIFY_SECRET_KEY, true,
IDENTIFY_BACK_URL);

// MBS billing library is initiated
define('SBBS_MBS_SERVICE_ID', 2);
define('SBBS_BACK_URL', 'http://'.$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']);
define('SBBS_SECRET_KEY', 'test');
$sbbs = new SBBS(SBBS_MBS_SERVICE_ID, null, SBBS_BACK_URL, SBBS_SECRET_KEY);

// The user is identified
if (!$identify->isIdentified()) {
    $identify->MakeIdentify(session_id(), array());
}

// The subscription is initiated
if (!isset($_GET['after_register'])){
          // Only the user who is not subscribed is subscribed
```

```
          if (!$identify->getSubscriberStatus()!='active') {
        // Here the user will be redirected to MBS for subscription
        $sbbs->registerRedirect(array(
            'session_id' => session_id(),
            'after_register' => 'yes',
        ), time());
          // Already subscribed user
    } else {
        echo "User, you have already susbscribed";
    }
// The end of subscription and correct data found by GET
} else if ($transaction=$sbbs->getRegisterInformation()) {
          // Successful subscription
    if ($transaction['subscriber_status']=='active' ) {
        echo 'The user has successfully subscribed';
    // Unsuccessful subscription
    } else {
        echo 'Unsuccessful subscription';
    }
// The end of registration, but GET data are incorrect
} else {
    echo "Subscription error: ".$sbbs->getError();
}

?>
```

Redirection to MBS for unregistration

The browsing user clicks on the link which redirects the user to the MBS unsubscription environment. The process is very similar to subscription, except for the question. The user is immediately removed from subscription and is returned to the service.

To unregister the WAP user, she/he must be redirected to MBS at http://bill.vero.lt/wap/unregister with formatted GET array parameters:

Name	Type	Mandatory	Description
s	int	yes	WAP service ID for subscription
si	string	no	The user's identifier in the subscription service (e.g. password). It is used, if the same user subscribes to the service more than once but uses different identifiers.
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction). The parameter is also used to avoid double billing, therefore, each time it must be unique for one service or at least one user. If you do not store billing events, it is possible to show the time of the link generation.
u	string	no	Encoded (urlencode) URL, whereto the user is redirected after the unsubscription. In the absence of this parameter, the user will be

			redirected to the service URL indicated during the service creation.
un	string	no	Encoded (urlencode) URL, if it is necessary to redirect the user to a different URL after the unsuccessful subscription. In the absence of this parameter, the user will be redirected to URL indicated in parameter *u*.
s1	string	yes	Security signature (lower security level)

The example of a generated address (URL) whereto the user is redirected (the security signature uses the password 'test'):

```
http://bill.vero.lt/wap/unregister?after_unregister=yes&s=2&si=&i=1255531284&u=http%3A
%2F%2Fmbs.vero.lt%2Fsimple
%2Fwap_unregister.php&s1=4da55a594963f371d2d0940a237d90545b1c27bf
```

Additional parameters may be added. Later, together with the billing data they will be added to the address, using which the user with the subscription data will return to the project. Note: parameters cannot have such names as 's', 'si', 'i', 'u', 'un', 's1', 's2', as these parameters are used for transferring the key information.

To avoid double billing, it is not enough to generate a billing link by clicking on a purchase button. In such a case, by clicking on the same button twice in a row, you will generate two different purchase requests and both of them will be paid for. We recommend making the purchase button a generated purchase link with the unique parameter *i*.

If you initiate WAP unsubscription via SOAP, detailed descriptions on how to initiate subscription events, verify subscription status, unsubscribe, etc. may be found in Annex 'Description of Remote Methods'

Return of the user to the external system after unregistration

Name	Type	Mandatory	Description
s	int	taip	WAP service ID for subscription
si	string	no	The user's identifier in the subscription service (e.g. password). It is used, if the same user subscribes to the service more than once but uses different identifiers.
sr	string	yes	The subscriber's key status describes the activity of the user in the subscription. For a list of possible values see States and statuses of subscribers
srs	string	yes	The subscriber's additional state which specifies why the billing (subscription, renewal, etc.) is unsuccessful. For a list of possible values see States and statuses of subscribers
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction).
t	int	yes	Unique identifier of the billing event (transaction) in the MBS system
a	int	no	Unique identifier of the charged user in the MBS system (account_id)
s1	string	yes	Security signature (lower security level)

s2	string	yes	Security signature (highest security level)

The user is redirected to the service address after the (un)successful subscription (the link includes the parameters requested by the external system when initiating the event). Thus, for instance (the security signature uses the password 'test'):

```
http://mbs.vero.lt/simple/wap_unregister.php?
after_unregister=yes&s=2&si=&sr=removed&srs=remove_wait&i=1256379623&t=774&a=1&s1=feba1
958c0e7adcb8a21bc2ab464bbce8b6b240c&s2=j0lxrpWjzZIDcMUS7fW2D99Re6bE7D%2FfWeDVxLltF4Y
%2B5XwX0BwpqgG%2BUlepbahXmp1yzoop9pS1QvE9SbrxDIoZh1Jp5kbMtJ%2FHt6Dvtn
%2FSQiNvxkXohTRekxR871AY9Ud2uoLXQUe9KhVqQQqwrjYR2RvdrLRol6z0L6J2ioE%3D
```


When the user returns to the service address, it is necessary to verify the security signature (parameter 's1' or 's2'), thus ensuring that the user is redirected from MBS.


When the user returns to the service address, it is necessary to verify the subscriber's status (parameter 'r1' or 'srs') and make ensure that the service is not provided in case of unsuccessful subscription.


Subscription report *register* is sent to the partner after a few seconds or minutes. However, some delays are possible.

Example of the WAP unregistration

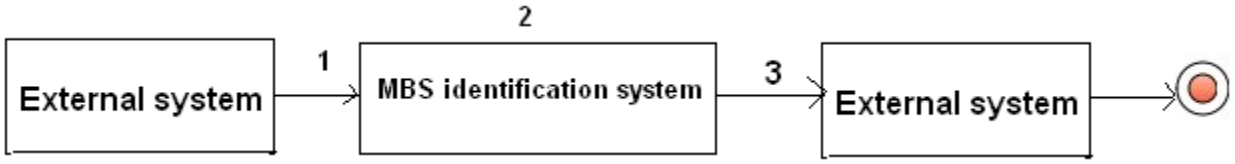
The MBS library is used in the example. For more information see 'Description of Remote Methods'.


If you do not want to use the MBS PHP library, you should implement some functions for your convenience, e.g. *signArrayLow()*, *verifyArrayHigh()*, *MakeIdentify()* and *_getUserInformation()*.

```
<?php
/**
 * Raw WAP unregistration example (using redirect)
 *
 * @see MBSLib/examples/simple/wap_unregister.php
 *
 * @package RawExamples
 * @author Valdas Petru.is <vpe@vero.lt>
 */

session_start();
include_once(dirname(__FILE__).'/../../lib/SBBS.class.php');

// SBBS billing library is initiated
define('SBBS_MBS_SERVICE_ID', 2);
define('SBBS_BACK_URL', 'http://'.$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']);
define('SBBS_SECRET_KEY', 'test');
$sbbs = new SBBS(SBBS_MBS_SERVICE_ID, null, SBBS_BACK_URL, SBBS_SECRET_KEY);
```

```
// Initiation of unsubscription
if (!isset($_GET['after_register'])) {
    $sbbs->unregisterRedirect(array(
        'after_unregister' => 'yes',
    ), time());
// end of subscription and correct data found by GET
} else if ($transaction=$sbbs->getUnregisterInformation()) {
          // Successful unsubscription
          if ($transaction['subscriber_status']=='removed' ) {
        echo 'The user is removed from the subscription';
    // Unsuccessful unsubscription
    } else {
        echo 'The user's unsubscription failed';
    }
// The end of unsubscription, but GET data are incorrect
} else {
          echo "Unsubscription error: ".$sbbs->getError();
}

?>
```

WAP Identification Services



Schema 10WAP identification stages

4. When the external system wants to identify a user, it redirects it to MBS (http://bill.vero.lt/identify) with the parameters specified in section "Redirection to MBS to identify a user";
5. MBS identifies a user;
6. A user is redirected back to the external system with the parameters specified in section "User's return to the external system after the identification".

When using the identification service, the user must be identified only once rather than during every request. For this purpose, there is a special parameter transfer mechanism that allows adding the parameter chosen (e.g.: session) to ensure that the user does not have to be re-identified. Services which identify the same user more than once will be automatically disabled.

If the MBS identification system is unable to identify the user, and the service is in the debug mode, the user will receive the identification error and will not be redirected to the project. If the user is not identified in the normal mode, it is returned to the project with 'a' parameter equal to '0' (no quotation marks).

Redirection to MBS for user identification

To identify a user, it has to be redirected to the MBS identification system at http://bill.vero.lt/identify with the following parameters:

Name	Type	Mandatory	Description
i	int	Yes	Identification service ID. It is used to indicate the external system that asks for identification.
si	string	No	Users identificator in subscription service (for example nickname). Used if user can be registered in same service more then once with different identificator.
t	string	No	Transaction. A parameter of the external system that indicates the identification event.
u	string	No	URL encoded to which a user has to be redirected after the identification. If there is no such parameter, a user is redirected to a service URL indicated during the creation of the identification service.
s1	string	Yes	Security signature (lower security level)

An example of an address formed with parameters where a user will redirected (as security signature we use „test"):

```
http://bill.vero.lt/identify?i=1&t=1255617369&u=http%253A%252F%252Fmbs.vero.lt
%252Fsimple
%252Fwap_identify.php&session_id=d9532e97acebdd356fc465df249b4bce&after_identify=yes&s1
=ed168dbd6ba4cefa723144dcf2839fa5a8fa5030
```

Any parameters may be added additionally. After a user's identification they together with the identification data will be added to the address that a user will use to return to the project. However, parameters cannot be named by 'i', 's', 's1', 's2', 't', 'ac', 'al', 'op', 'a', 'oi', 'om', 'si', 'sr', 'srs', 'sid', 'u', 'k', as they are used to transfer the main information.

Return of the user to the external system after identification

Name	Type	Mandatory	Description
i	int	yes	Identification service ID. It is used to indicate the external system that asks for identification.
t	string	no	Transaction. A parameter of the external system that indicates the identification event.
ac	string	no	User country is indicated in ISO 3166-1 format (Alpha-2). The configuration of the identification service determines whether the parameter will be sent.
al	string	no	User language is indicated in ISO-639-1 format (the Estonian language – "ET"). The configuration of the identification service determines whether the parameter will be sent.
op	string	no	Operator, e.g. omnitel_lt, bite_lv, era_pl, etc (Operator codes). The configuration of the identification service determines whether the parameter will be sent.
a	int	yes	User's unique identifier in the MBS system.
oi	string	no	Operator's identifier. It is unique in every operator's system. It may be a telephone number or, in Bitė's case, bite-account-Id. The configuration of the identification service determines whether the parameter will be sent.
om	string	no	User's telephone number (MSISDN). It is determined not to all operators.
ot	string	no	User's payment type (prepaid, postpaid, testing, unknown). It is determined not to all operators.
si	string	no	User's identifier in a subscription service (e.g. user name). It is used, if the same user but with different identifiers can subscribe the service.
sr	string	no	Subscriber's main status which describes the user's activity in the subscription. For a list of possible values please see 'States and Statuses of Subscribers'.
srs	string	no	Subscriber's additional state which establishes the reason for the failed payment (registration, renewal, etc.). For a list of possible values please see 'States and Statuses of Subscribers'.
s1	string	yes	Security signature (lower security level)
s2	string	yes	Security signature (highest security level)

The user is redirected to the service address after the (un)successful identification (the link includes the parameters requested by the external system when initiating the event). Thus, for instance (the security signature uses the password ‘test’):

```
http://mbs.vero.lt/simple/wap_identify.php?
i=1&t=1255617369&ac=LT&al=LT&op=bite_lt&a=1&si=&sr=removed&srs=subscribe_cancel&session
_id=d9532e97acebdd356fc465df249b4bce&after_identify=yes&s1=0bd58d0c4ac6d60d84f724e93fcd
eede091c1171&s2=nAMQm9hAr9ExY2LYIqG2KR9V8vARbuiACtNUOMvJf6RbJ%2Bj
%2B4SYO3gkTKOF3tiscfDkkdc6GHVOekcBUpYTr6wDLzN2zctI4Nz3ZxA3qkyXjm4U3Rec4GjBHA87Jj43SC
%2FCLBGvICgpxELUoDGITxFxkE1yxDrgHjiph4HYt8kw%3D
```


When the user returns to the service address, it is necessary to verify the security signature (parameter ‘s1’ or ‘s2’), thus ensuring that the user is redirected from MBS.


If the MBS identification system fails to identify the mobile user, on returning to the project the parameter ‘a’ will be equal to ‘0’.

Example

The MBS PHP library is used in the example. For more information see ‘Description of Remote Methods’.


If you do not want to use the MBS PHP library, you should implement some functions for your convenience, e.g. *signArrayLow()*, *verifyArrayHigh()*, *MakeIdentify()* and *_getUserInformation()*.

```
<?php
/**
 * Raw identify example
 *
 * @see MBSLib/examples/simple/wap_identify.php
 *
 * @package RawExamples
 * @author Šarūnas Davalga <sd@vero.lt>
 */

session_start();
include_once(dirname(__FILE__).'/../../lib/Identify.class.php');

// Initiation of the MBS identification library
define('IDENTIFY_SERVICE_ID', 1);
define('IDENTIFY_BACK_URL', 'http://'.$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']);
define('IDENTIFY_SECRET_KEY', 'test');
$identify = new Identify(IDENTIFY_SERVICE_ID, IDENTIFY_SECRET_KEY, true, 'http://mb-
s.vero.lt/simple/wap_identify.php');

// Check if the user returned after the identification
if (!$identify->isIdentified()) {
          // the user will be redirected to  MBS for identification
          $identify_url = $identify->MakeIdentify(session_id());
} else {
          // Ar pavyko identifikuoti vartotoją ir nėra klaidų
```

```
        if ( $identify->getAccount() != 0 && !$identify->isError()) {
                session_id($identify->getTransaction());
                echo '<pre>';
                print_r($identify->getUserInfo());
                echo '</pre>';
        } else {
                // If the error occurs, the error message is printed out
                echo $identify->getError();
                // The user is prepared for a re-identification
                $identify->unsetUserData();
        }
}

?>
```

Partial data identification

The MBS identification is usually used to determine the precise data of the browsing user: unique user of the operator, user's MSISDN (telephone number), language, etc. MBS may be also used for the partial data identification:

- Country of the browsing user;
- Language of the browsing user;
- If the user may not be identified, MBS tries to identify at least the user's operator. The operator's value is not returned only if the user does not use a mobile operator (WEB browser) or MBS does not support this operator.

The coverage of MBS countries and operators is constantly expanding, identification quality increasing, the latest operators' enhancements and bug fixes being updated (e.g. support of the operator's browsers). The country is always identified, however, it is not always possible to identify the user's language and operator. A full list of support operators is here.

Operator gateway IP list

It is possible to identify operator by WAP user's IP address. To get a list of operators IP you can use the following URL that returns last update time followed by IP list `http://bill.vero.lt/identify/ip`

Also operator can be identified by hostname. Hostname rules ar dislpayed in MySQL Regex format. To get a list of operators hostname rules, use: `http://bill.vero.lt/identify/host`

Name	Type	Mandatory	Description
operator	string	yes	IP's will be shown for this operator, e.g. omnitel_lt, bite_lv, era_pl, etc. (Operator codes)

format	string	no	Defines a format of list: • list – returns IP list • short – returns IP range list • last_update – returns date of last update

Examples:

```
http://bill.vero.lt/identify/ip?operator=tele2_lt&format=short
2009-10-22 09:57:26

83.178.0.220
83.178.2.158
83.178.42.4
83.178.56.2-83.178.57.254
```

```
http://bill.vero.lt/identify/host?operator=tele2_lt&format=list
2010-05-05 22:21:32

\.swipnet\.se$
\.swip\.net$
\.tele2\.
```

Tele2 in all countries (tele2_lt, tele2_lv, tele2_ee etc.) use the same IP addresses. To fully identify Tele2 user use Redirection to MBS for user identification

If you have an agreement with the operator

This section is for you, if:

- your project requires the identification without redirection (not all telephone devices support uninterrupted redirections);
- operators (or one operator) decided to enable the identification data transmission for your project/server.

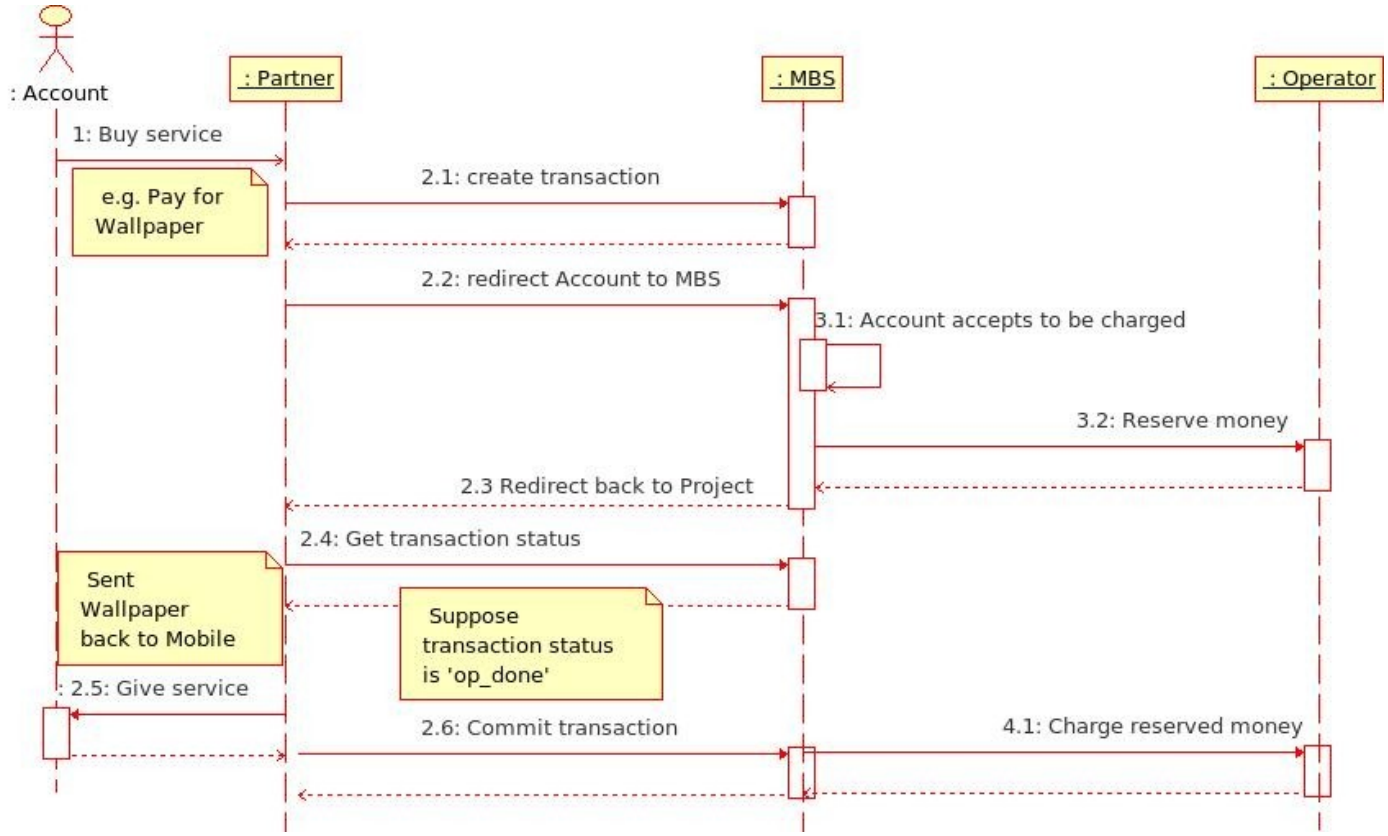
In such a case, MBS has functionality which enables identifying the user without redirection, i.e. remote identification. This functionality is used with a help of the MBS library methods *MakeIdentify* or *MakeIdentifyRemote*. For more information please the library configuration MBS_IDENTIFY_TYPE. When changing this configuration, the solution will operate with the standard identification example.

Why it is better to use this type of identification rather than to develop your own identification system:

- **Installation of the operator's updates:** we follow all changes, updates and enhancements of the operator's systems, collect the latest operator's IP addresses. All updates, as soon as possible, are installed in one system and instantly reach all projects used by it.
- **Support of identification errors:** all identification results are always monitored. If the operator's system is not working properly, the errors are corrected and the operator is informed. We have a great number of identification operations, therefore, it is easier to ask operators to solve identification problems, improve their quality;
- **Keeping of statistics:** MBS stores all (un)successfully identified users. If required, we can always provide you with the statistics of users who visited your project.
- **Improvement of the identification quality:** the MBS system is being constantly updated, its quality improved; the support of new operators and their technologies (e.g. WAP browsers) enhanced.

WAP Billing Services

The service is for single billing of the user in the WAP environment. A browsing user clicks on a link which redirects her/him to the MBS billing environment. The user is asked if she/he agrees to pay for the service. If yes, the user is charged, if no – the transaction is canceled. Either way, the user is returned to the WAP site where a service paid for is provided or the error displayed. Please see the scheme bellow:



Schema 11WAP billing stages

- *The project* redirects *the user* to *MBS* with the payment data (2)
- *MBS* work begins (3)
  - (*not mandatory*) *the user* confirms that she/he wants to receive a service (3.1)
  - *MBS* reserves the money in the user's account (3.1)
  - *MBS* returns the user to the project (3.3).
- *The project* makes sure the transaction has been paid for (2.4) and provides the service to the user (2.5)
- (*not mandatory*) *The project* confirms or declines the payment (2.6)

In case of the operator Omnitel Lietuva, during the billing procedure, the user is sent to the waiting page with a message '*The payment is in process. Once you receive the SMS message, click on this link: to continue>*'. A WapPush payment message is sent to the user, indicating the name of the

service and a link to the partner's service. Hence, even if the user leaves a WAP browser, she/he will be able to continue downloading the content by clicking on the link received by the WapPush message.

If partner receives payment errors, but error is not processed correctly (HTTP status is not 200), operator Omnitel Lietuva payments will fail!!!

Redirection to MBS for user billing

To charge the WAP user, she/he must be redirected to MBS at http://bill.vero.lt/wap/bill with formatted GET arrays:

Name	Type	Mandatory	Description
s	int	yes	Billing service ID
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction). The parameter is also used to avoid double billing, therefore, each time it must unique for one service or at least one user. If you do not store billing events, it is possible to show the time of the link generation.
p	int	no	Billing price in cents. It is required when one service has several prices.
pc	string	no	Billing currency (established by ISO 4217). It is required when specifying the price in parameter *p*.
u	string	no	Encoded (urlencode) URL, whereto the user is redirected after the billing. In the absence of this parameter, the user will be redirected to the service URL indicated during the service creation.
un	string	no	Encoded (urlencode) URL, if it is necessary to redirect the user to a different URL after the unsuccessful billing. In the absence of this parameter, the user will be redirected to URL indicated in parameter *u*.
s1	string	yes	Security signature (lower security level)

The example of a generated address (URL) whereto the user will be redirected (the security signature uses the password 'test'):

```
http://bill.vero.lt/wap/bill?
act=mbs.donate&session_id=d1563ade68224d5e4f9ea26951d67215&s=1&i=&p=30&pc=LTL&u=http%3A
%2F%2Fmbs.vero.lt%2Findex.php&s1=609a35c5db0c6ae81f57525d3622f32ed4d0ee96
```

Additional parameters may be added. Later, together with the billing data they will be added to the address, using which the user will return to the project. Note: parameters cannot have such names as 's', 'i', 'p', 'pc', 'u', 'un', 's1', 's2', as these parameters are used for transferring the key information.

To avoid double billing, it is not enough to generate a billing link by clicking on a purchase button. In such a case, by clicking on the same button twice in a row, you will generate two different purchase requests and both of them will be paid for. We recommend making the purchase button a generated purchase link with the unique parameter *i*.

If you initiate WAP billing via SOAP, detailed descriptions how to initiate billing events, verify billing status, confirm or cancel the amount charged, etc. may be found in Annex 'Description of Remote Methods'

Redirection of the user to the external system after payment

Name	Type	Mandatory	Description
s	int	yes	Billing service ID
r	string	yes	The main status of the billing event (transaction), which specifies if the billing is successful. For a list of possible values see 'Transaction Status'.
rs	string	yes	The additional state of the billing event (transaction) which specifies why the billing is unsuccessful. For a list of possible values see 'Transaction Status'.
i	string	yes	The identifier of the external system which uniquely describes the payment in process (transaction).
t	int	yes	Unique identifier of the billing event (transaction) in the MBS system.
a	int	yes	Unique identifier of the charged user in the MBS system (account_id).
s1	string	yes	Security signature (lower security level)
s2	string	yes	Security signature (highest security level)

The user is redirected to the service address after the (un)successful billing (the link includes the parameters requested by the external system when initiating the event). Thus, for instance (the security signature uses the password 'test'):

```
http://mbs.vero.lt/simple/wap_bill.php?
session=df338a8a463fa8e8ce8d31e2b5bca6ed&after_bill=yes&s=1&r=commit&rs=op_done&i=12555
25204&t=2331137&a=1&s1=e728b3286b956729d969cb8d78b635ceb7753ea8&s2=ZBxZg02d82VLICXo
%2BznI8ZQjgkIrC%2FWXCEHYb%2FAL%2FTO8v2BXZafk%2FXMY9iJ%2FXjzaOSQxelW%2FMlau
%2FJgcNl7IXDLM4gxrwL7jgFxKn0rn2Z4mJ%2FauKWNG1itbTw37YaEZdZVJg0wBaUbcuiWBy
%2B3ldle5RZRGKJV9qqgE%2FkN73%2Fg%3D
```


When the user returns to the service address, it is necessary to verify the security signature (parameter 's1' or 's2'), thus ensuring that the user is redirected from MBS.


When the user returns to the service address, it is necessary to verify the billing status (parameter 'r1' or 'rs') and make ensure that the service is not provided for unsuccessful transactions.


When the user returns to the service address, you can check if the browsing user is the same user who was charged (parameter 'a'). This way you make sure that you do not provide the service when the return of someone else's transaction is imitated.

Example

The example (WAP billing, the results of which are displayed on the screen) is implemented by using the MBS PHP library.

```
<?php
/**
 * Raw WAP billing example (using redirect)
 *
 * @see MBSLib/examples/simple/wap_bill.php
```

```
 *
 * @package RawExamples
 * @author Karolis Tamutis <kt@vero.lt>
 * @author Valdas Petrulis <vpe@vero.lt>
 */

session_start();
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');

// MBS billing library is initiated
define('MBS_SERVICE_ID', 1);
define('MBS_BACK_URL', 'http://'.$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF']);
define('MBS_SECRET_KEY', 'test');
$mbs = new MBS(MBS_SERVICE_ID, null, MBS_BACK_URL, MBS_SECRET_KEY);

// Billing initiation
if (!isset($_GET['after_bill'])) {
    // The user is sent to MBS
    $mbs->BillRedirect(array(
                'session_id' => session_id(),
                'after_bill' => 'yes',
            ), time());
// End of billing and correct data are found in GET array
} else if ($transaction=$mbs->getBillInformation()) {
           // Successful transaction
           if ($transaction['status']=='commit') {
                       /** ... the service is provided here ... **/
                       $paslauga_suteikta = true;

                       // Billing confirmation via SOAP (not mandatory)
                       if ($paslauga_suteikta) {
                                   $mbs-
>commitTransaction($transaction['transaction_id']);
                                   echo "Successful service";
                       // Billing rollback via SOAP (not mandatory)
                       } else {
                                   $mbs-
>rollBackTransaction($transaction['transaction_id']);
             echo "Unsuccessful service";
                       }
           // Unpaid transaction
           } else {
                       echo "Unpaid transaction. ".$mbs->getStatusDescription($transac-
tion['state']);
           }
// End of billing, incorrect data in GET array
} else {
           echo "Billing error: ".$mbs->getError();
}
?>
```

SMS/WapPush Sending Services

The service is used for sending different information messages from the partner's software to mobile users. Thus, for instance:

- Sending reminders about profiles on a community site;
- Sending login authorisation to the user;
- Mass SMS/WapPush – sending advertisements (see SMS Mass Sending Services);


You can also use your system-to-system login to connect to the graphical user interface http://smscomm.vero.lt/. You can use manual sending, monitor statistics, etc.


It is recommended to order a separate sending login for every project. In this way you will be able to differentiate the flow of sent messages.


By default, if the SMS message to be sent is longer than 160 symbols, the MBS shortens it up to 160 symbols.


All SMS messages or WapPush symbols are replaced into Latin equivalents by default (ž -> z; ū,ų -> u, etc.).

Query format

The partner sends an HTTP query to http://smscomm.vero.lt/partner/sms.php. Parameters must be URL-encoded and in UTF-8 encoding. Data are presented by HTTP GET method:

Title	Description	Example
username	Username	demo
password	Login password	demo
action	Action to execute, possible values: • *send* – message sending (default value); • *validate* – message check for errors, sending not performed;	send
sender	Sender phone number. This function is turned on by individual agreement, then all possible values should be listed	VeroDemo
phone	Receiver's telephone number (MSISDN) in the international format "370xxxyyyyy"	37065659515
sms	SMS text to be sent (in case of WapPush, the text is the link heading)	Demo SMS API message
url	(Optional) In case of WapPush, a link of WapPush to be sent	http://projektas.lt/puslapis?parametras=reiksme
ident	(Optional) External system unique identifier e.g. published advertisment unique number	LT123

type	(Optional) Type of sending (SMS, WapPush), by default SMS	SMS
service_id	(Optional) MBS service ID, you will see seperated statictics of each service sending costs	54321
operator	(Optional) To be indicated only if the partner knows the recipient's operator. In such a case, the message is sent at the rate that is most favourable to the operator. Format: tele2_lt, bite_lt, tele2_lv, etc.	bite_lt
send_date	(Optional) Message sending date. To be indicated only if the partner wants to send the message later. If not indicated, the message is sent immediately.	2009-11-13 14:00:00
drlurl	(Optional) URL address, whereto the SMS delivery results should be sent. To be indicated only if the partner knows how to process delivery reports.	http://projektas.lt/zinutesPristatymas.php
priority	(Optional) Message priority. Higher priority is used if you need to send important messages (e.g. user registration approval) faster than other messages in queue (e.g. mass advertising). • *2* – Soon; • *3* – Rather soon; • *4* – At the occasion; • *5* – Unurgent; • *6* – It does not matter;	3
charset	(Optional) Sending text rule (charset): • *lisp* – Lisped symbols; • *windows-1252* – Latin sybbols; • *windows-1257* – Baltic country symbols; • *windows-1251* – Cyrillic symbols; • *utf-8* – Any symbols; If not indicated, service configuration will be used. MBS will lisp given text or will detect most suitable charset for given text.	lisp

When HTTP GET query is executed, the MBS replies in a text line, the format of which is {RESULT} {RESULT _DESCRIPTION}. A list of possible results:

RESULT	Description
OK {SMS_ID}	The MBS accepted the SMS order and provides it with a unique identifier in the MBS – *SMS_ID* (e.g. 12345)
ERROR {ERROR _DESCRIPTION }	The SMS order failed

Example of the HTTP query:

```
http://smscomm.vero.lt/partner/sms.php?
username=demo&password=demo&sender=VeroDemo&phone=37065659515&type=SMS&sms=Demo+SMS+API
+message
```

Do not forget to encode the necessary parameters (URL-encode). For URL-encode encoding in PHP language functions *http_build_query()* or *urlencode()* is used.

Example

The example (performs message sending to user and result is displayed on the screen).

If you are using PHP programming language, you can use MBS PHP library class SMSComm

```
<?php
/**
 * Raw SMS sending example
 *
 * @see MBSLib/examples/simple/send_sms.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// Žinučių siuntimo konfigūracija
define('SMSCOMM_SMS_URL', 'http://smscomm.vero.lt/partner/sms.php');
define('SMSCOMM_USERNAME', 'demo');
define('SMSCOMM_PASSWORD', 'demo');

$url = SMSCOMM_SMS_URL.'?'.http_build_query(array(
    'username'  => SMSCOMM_USERNAME,
    'password'  => SMSCOMM_PASSWORD,
    'sender'    => 'VeroDemo',
    'phone'     => '37065659515',
    'sms'       => 'Demo SMS API message',
));

$result = trim(file_get_contents($url));
if( $result=='OK' ) {
          echo 'SMS sent';
} else {
          echo 'Error in SMS: ' . $result;
}

?>
```

Delivery reports

If a message to be sent includes a parameter *dlrurl*, message delivery reports will be sent to the URL address indicated in the parameter. Parameters indicated in the HTTP GET query:

Title	Description	Example

msg_id	Unique identifier of the message sent in the MBS – *SMS_ID*. It is received when ordering the service.	12345
ident	(Optional) External system unique identifier e.g. published advertisment unique number	LT123
from	Sender phone number that was used	1679
to	Receiver's telephone number (MSISDN) in the international format "370xxxyyyyy"	37065659515
operator	Receiver's operator that was detected tele2_lt, bite_lt, tele2_lv ir etc	bite_lt
charset	Which sending text rule was	lisp
length	Number of packets in message	1
delivery	Digital message delivery status (see below)	1
status	Text message deliver status (see below)	DELIVERED

Example of the HTTP query:

```
http://projektas.lt/zinutesPristatymas.php?
msgId=12345&drlId=1&phone=37065659515&info=DELIVERED
```

Possible message delivery statuses:

Digital format	Text format	Explanation
16	REJECTED	The system refused to send the message.
8	SUBMITTED	The system delivered the message to the operator.
4	BUFFERED	Message queued to be sent later
2	NOT_DELIVERED	The message was not delivered to the recipient.
1	DELIVERED	The message was delivered to the recipient.

Remember, the delivery report is sent when the message status changes in the MBS. As a result, you will receive several delivery messages, e.g. if successful – SUBMITTED, later – DELIVERED.

SMS Mass Sending Services

The service is used for sending big amounts of information messages (SMS/WapPush) from the partner's software to mobile users. See several examples, why it is useful to use mass sending instead of single sending service:

- If your system sends messages big amount messages e.g. each morning 09:00 sends 1000 messages to all registered users;
- Need of high bandwidth (sending speed), 1000 messages ordering happens quickly and sending is performed straightaway;
- Partner's and MBS servers is not loaded because instead of 1000 requests only one order is performed for overall packet;
- If your system sends same message text to big amount of receivers e.g. notification for all news subscribers;
- If you need additional features: watch sending and delivery statistics, send delayed messages, edit, stop, delete wrongly formatted packets;

Query format

Partner system sends an HTTP query to URL http://smscomm.vero.lt/partner/sms_packet.php, data are presented:

Title	Description	Example
username	Username	demo
password	Login password	demo
action	Action to execute, possible values: • *validate_packet* – mass message check for errors, sending not performed; • *send_packet* – mass message sending; • *get_packet* – mass message sending, delivery statistics; • *edit* – to edit unsent messages; • *stop* – to stop messages sending; • *delete* – to remove unsent messages;	send_packet
HTTP POST	Action detailed parameters, more about their values read in specific action description. Parameters structure is encoded in JSON format and given by HTTP POST method.	array ('ident' => 1323202186, 'sender' => 'VeroDemo', 'sms' => 'Demo SMS API message', 'messages' =>

		array (1 => array ('phone' => '37065659515',), 2 => array ('phone' => '37060042751', 'sms' => 'Dear John. Demo SMS API message',),),)

When HTTP query is executed, the MBS replies in JSON response, formatted by JSEND specification (see http://labs.omniti.com/labs/jsend/wiki). It strictly defines 3 response types:

- *success* – successful event;
- *fail* – failure event (e.g. wrong data);
- *error* – error event (an exception was thrown, DB error, ...);

Example of the HTTP query:

```
http://smscomm.vero.lt/partner/sms_packet.php?
username=demo&password=demo&action=send_packet
```

HTTP POST example:

```
{"ident":1323203033,"type":"SMS","sender":"VeroDemo","sms":"Demo SMS API
message","messages":{"1":{"phone":"37065659515"},"2":{"phone":"37060042751","sms":"Dear
John. Demo SMS API message"}}}
```

HTTP response example (successfull event):

```
{"status":"success","data":{"sent":{"1":11021391,"2":11021392},"errors":[]}}
```

HTTP response example (failure event):

```
{"status":"fail","data":{"sent":[],"errors":{"1":{"field":"phone","field_error":"Wrong
MSISDN format"}}}}
```

HTTP response example (error event):

```
{"status":"error","message":"Messages list is empty or not an array"}
```

validate_packet, send_packet

Action *validate_packet* is dedicated to check mass message for possible errors, sending not performed. Action *send_packet* – mass message sending.

Request parameters:

Title	Description	Example
messages	Receiver's list - associative array, which contains	array(

	key-value pairs: • value must have required parameter "phone" - receiver's telephone number (MSISDN) in the international format "370xxxyyyyy"; • key contains receiver unique identifier in list. Used later to relate with packet errors • value can contain personal parameters of that receiver's message, which are the same as in sending single message, e.g. by giving parameter "sms" different message text can be formed for specific receiver;	'1' => array('phone' => '37065659515',), '2' => array('phone' => ' 37060042751', 'sms' => ' Dear John. Demo SMS API message));
skip_errors	(Optional) How to handle packet messages with errors (possible only in *send_packet* action): *0* – if packet contains errors, sending not performed (default value); *1* – send messages in any case, errors will be skipped;	0
sender	(Optional) Sender phone number. This function is turned on by individual agreement, then all possible values should be listed	VeroDemo
sms	(Optional) SMS text to be sent (in case of WapPush, the text is the link heading)	Demo SMS API message
ident	(Optional) External system unique identifier e.g. mass message identifier	news-reminder-12345
...	(Optional) Parameters the same as in sending single message can be used	

Response parameters:

Status	Description	Example
success	All messages or part of them successfully sent. Returns: • *sent* – succesfully accepted messages provided with a unique identifier in the MBS – *SMS_ID*; • *errors* – rejected messages and their errors (then using parameter *skip_errors=1*);	array('status' => 'success', 'data' => array('sent' => array('1' => 12345, '2' => 12346,), 'errors' => array()))
fail	Messages was not sent because of wrong data given. Returns: • *errors* – rejected messages and their errors; • *field* – which field contains error; • *field_error* – error description;	array('status' => 'success', 'data' => array('sent' => array(), 'errors' => array('3' => array('field' => 'phone', 'field_error' => 'Wrong

		msisdn format'))))
error	Critical or unexpected error of implementation	array('status' => 'error', 'message' => 'Fatal DB error',)

get_packet

Action *get_packet* is dedicated to retrieve mass message sending, delivery statistics.

Request parameters:

Title	Description	Example
ident	Indicate which messages to retrieve, by external system unique identifier e.g. mass message identifier	news-reminder-12345

Response parameters:

Status	Description	Example
success	Returns mass message information: • *ident* – external system identifier of messages; • *stats_overal* – amount of total messages; • *stats_sent* – amount of already sent messages; • *stats_delivered* – amount of messages delivered to users; • *stats_length* – amount of chunks to be paid (1 packet contains 160 characters);	array('status' => 'success', 'data' => array('ident' => 'news-reminder-12345', 'stats_overal' => '100', 'stats_sent' => '100', 'stats_delivered' => '87', 'stats_length' => '87',))
error	Critical or unexpected error of implementation	array('status' => 'error', 'message' => 'Fatal DB error',)

edit

Action *edit* is dedicated to edit unsent messages (one or overal packet).

Remember, edit actions (*edit*, *stop*, *delete*) is executed only on unsent messages. Sent messages stays untouched.

Request parameters:

Title	Description	Example

ident	Indicate which messages to edit, by external system unique identifier e.g. mass message identifier	news-reminder-12345
message_id	(Optional) Indicate which messages to edit, by unique MBS identifier or array of them	array(12345, 12346, 12347)
sender	(Optional) If want to edit sender phone number	VeroDemo
sms	(Optional) If want to edit SMS text to be sent (in case of WapPush, the text is the link heading)	Demo SMS API message
new_ident	(Optional) If want to edit external system unique identifier e.g. edited messages can be formed to a new packet	news-reminder-12346
...	(Optional) Possible to edit all other parameters used in sending single message	

Response parameters:

Status	Description	Example
success	Messages successfully edited. Returns: • *edited* – amount of edited messages;	array('status' => 'success', 'data' => array('edited' => 125,))
fail	Messages was not edited because of wrong data data given. Returns: • *field* – which field contains error; • *field_error* – error description;	array('status' => 'fail', 'data' => array('field' => 'message', 'field_error' => 'Empty sending text',))
error	Critical or unexpected error of implementation	array('status' => 'error', 'message' => 'Fatal DB error',)

stop

Action *stop* is dedicated to stop messages sending for unlimited time period (one or overal packet).

Remember, edit actions (*edit*, *stop*, *delete*) is executed only on unsent messages. Sent messages stays untouched.

Request parameters:

Title	Description	Example
ident	Indicate which messages to stop, by external system unique identifier e.g. mass message identifier	news-reminder-12345
message_id	(Optional) Indicate which messages to stop, by	array(12345, 12346, 12347)

	unique MBS identifier or array of them	
new_ident	(Optional) If want to edit external system unique identifier while stoping e.g. stopped messages can be formed to a new packet	news-reminder-12346

Response parameters:

Status	Description	Example
success	Messages successfully stopped. Returns: • *edited* – amount of stopped messages;	array('status' => 'success', 'data' => array('edited' => 125,))
error	Critical or unexpected error of implementation	array('status' => 'error', 'message' => 'Fatal DB error',)

delete

Action *delete* is dedicated to unrecoverably remove unsent messages (one or overal packet).

Remember, edit actions (*edit*, *stop*, *delete*) is executed only on unsent messages. Sent messages stays untouched.

Request parameters:

Title	Description	Example
ident	Indicate which messages to delete, by external system unique identifier e.g. mass message identifier	news-reminder-12345
message_id	(Optional) Indicate which messages to delete, by unique MBS identifier or array of them	array(12345, 12346, 12347)

Response parameters:

Status	Description	Example
success	Messages successfully deleted. Returns: • *deleted* – amount of deleted messages;	array('status' => 'success', 'data' => array('deleted' => 125,))
error	Critical or unexpected error of implementation	array('status' => 'error', 'message' => 'Fatal DB error',)

Example

The example (performs mass message sending to several users and result is displayed on the screen) is implemented by using the MBS PHP library.

If you do not want to use the MBS PHP library, you should implement some functions for your convenience, e.g. *sendPacket()*, *_callPacketAction()*, *_sendRequest()*

```
<?php
/**
 * Raw mass SMS sending example
 *
 * @see MBSLib/examples/simple/send_sms_packet.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

include_once(dirname(__FILE__).'/../../lib/SMSComm.class.php');

try {
    // Init SMS API client with wanted configuration
    define('SMSCOMM_USERNAME', 'demo');
    define('SMSCOMM_PASSWORD', 'demo');
    $client = new SMSComm(SMSCOMM_USERNAME, SMSCOMM_PASSWORD);
    $client->setDebug(true);

    // Send mass SMS
    $ident = time();
    $messages = array(
        1 => array('phone' => '37065659515'),
        2 => array('phone' => '37060042751', 'sms' => 'Dear John. Demo SMS API
message'),
    );
    $result = $client->sendPacket($messages, false, 'VeroDemo', 'Demo SMS API message',
null, $ident);
    if ($result['status']=='success') {
        echo 'Sent '. count($result['data']['sent']) .' SMS messages, '.
count($result['data']['errors']) .' errors';
    } else {
        echo 'Errors in SMS messages: '.var_export($result['data']['errors'], true);
    }

    // Watch sending/delivery statistics
    $stats = $client->getPacket($ident);
    echo 'Packet information: '. var_export($stats, true);
} catch(Exception $e) {
    echo "Error in SMS API: " . $e;
}

?>
```

Annexes

Annex A – Security

It is crucial to protect the communication between MBS and the external system from the intervention of the third parties, data imitation, falsification and other dangers. To this end, MBS signs all requests sent to the external system and the external system responses only to properly signed requests. Similarly, the external system has to sign the requests sent so that MBS would accept them.

For your convenience, we let you choose the method for security checks. However, we highly recommend using the (highest security level) signature verification.

Security signature (highest security level)

OpenSSL algorithm (http://www.openssl.org/) is used for signing, where one party signs a request by a private key when sending it to the other party. The other party having a respective public key (half of the pair) may check if the request has not been changed.

Currently, this method is used only if the request is sent and signed by MBS. It is not used in cases where the external system sends the request. In such a case, every partner should have their own private keys to sign.

All MBS subsystem uses parameter *s2* for this method. Hence, if the external system wants to check if the data were transferred by MBS, it generates a signature for a line consisting of all data sent by the HTTP request (except for parameters *s1* and *s2*). Then MBS, using its public key, checks if the result received is the same as in sent parameter *s2*. Verification requires the MBS public key:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDRAYP2DXp/h59XAOGJ8VXTT5Pf
g4o3OAauQc2wcxfkHxrmwCpk0mCSjKE/nYcsQMuStrbBtHKmcu9IVZ1GULUWiEBt
/YO2bkhSXhGqmlpUR1J05+C/CKECpqfVZW6gFUH60O01AFinbinIkXVLeUwbtTIb
y9dSQCg2tivv6RTQDwIDAQAB
-----END PUBLIC KEY-----
```

Example of the request signed by MBS:

```
http://mbs.vero.lt/simple/verify_high.php?
i=1&t=1255617369&ac=LT&al=LT&op=bite_lt&a=1&si=&sr=removed&srs=subscribe_cancel&session
_id=d9532e97acebdd356fc465df249b4bce&after_identify=yes&s1=0bd58d0c4ac6d60d84f724e93fcd
eede091c1171&s2=nAMQm9hAr9ExY2LYIqG2KR9V8vARbuiACtNUOMvJf6RbJ%2Bj
%2B4SYO3gkTKOF3tiscfDkkdc6GHVOekcBUpYTr6wDLzN2zctI4Nz3ZxA3qkyXjm4U3Rec4GjBHA87Jj43SC
%2FCLBGvICgpxELUoDGITxFxkE1yxDrgHjiph4HYt8kw%3D
```

Example of the request verification implemented using the MBS PHP library (notification on whether it is the MBS data is displayed on the screen):

If you do not want to use the MBS PHP library, you should implement your own *verifyArrayHigh()* function.

```
<?php
/**
 * Security check example (using OpenSSL public key)
 *
 * @see MBSLib/examples/simple/verify_high.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// MBS function library is initiated
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');
$mbs_base = new MBSBase();

// GET array signed by MBS
if( $mbs_base->verifyArrayHigh($_GET) ) {
          echo 'Reliable MBS data';
// Falsified GET array
} else {
          echo 'Incorrectly signed data';
}

?>
```

Security signature (lower security level)

SHA1 algorithm (http://www.faqs.org/rfcs/rfc3174) is used for signing, where one party sends a request to the other party signed using a secret key known to both parties (salt). The other party knowing this key may check if the request has not been changed.

This method is used in both cases – where a request is sent and signed by MBS and where a request is sent by the external system.

The parties must agree on a secret key (salt) when creating a service. Its length is in proportion with the breaking possibility, therefore, it should not be shorter than of 15 symbols.

All MBS subsystems use parameter *s1* for this method. Hence, if the external system wants to check if the data were transferred by MBS, it generates a signature for a line consisting of all data sent by the HTTP request (except for parameters *s1* and *s2*) and adds a secret key (salt). Then, MBS encrypts the received line by using SHA1 algorithm and checks if the result received is the same as in sent parameter *s1*.

The example of a request signed by MBS (the security signature uses the password ‘test’):

```
http://mbs.vero.lt/simple/verify_low.php?
i=1&t=1255617369&ac=LT&al=LT&op=bite_lt&a=1&si=&sr=removed&srs=subscribe_cancel&session
_id=d9532e97acebdd356fc465df249b4bce&after_identify=yes&s1=0bd58d0c4ac6d60d84f724e93fcd
eede091c1171&s2=nAMQm9hAr9ExY2LYIqG2KR9V8vARbuiACtNUOMvJf6RbJ%2Bj
%2B4SYO3gkTKOF3tiscfDkkdc6GHVOekcBUpYTr6wDLzN2zctI4Nz3ZxA3qkyXjm4U3Rec4GjBHA87Jj43SC
%2FCLBGvICgpxELUoDGITxFxkE1yxDrgHjiph4HYt8kw%3D
```

Example of the request verification implemented using the MBS PHP library (notification on whether it is the MBS data is displayed on the screen):

If you do not want to use the MBS PHP library, you should implement your own *verifyArrayLow()* function.

```
<?php
/**
 * Security check example (using secret key)
 *
 * @see MBSLib/examples/simple/verify_low.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// Library of MBS functions is initiated
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');
$mbs_base = new MBSBase();
$mbs_base->setSecretKey('test');

// GET array sighed by MBS
if( $mbs_base->verifyArrayLow($_GET) ) {
          echo 'Reliable MBS data';
// Falsified GET array
} else {
          echo 'Incorrect signed data';
}

?>
```

Example of the request verification implemented using the MBS PHP library (notification on whether it is the MBS data is displayed on the screen):

If you do not want to use the MBS PHP library, you should implement your own *signUrlLow()* function.

```
<?php
/**
 * Security sign example (using secret key)
 *
 * @see MBSLib/examples/simple/sign_low.php
 *
 * @package RawExamples
 * @author Valdas Petrulis <vpe@vero.lt>
 */

// ILibrary of MBS functions is initiated
```

```
include_once(dirname(__FILE__).'/../../lib/MBS.class.php');
$mbs_base = new MBSBase();
$mbs_base->setSecretKey('test');

echo $mbs_base->signUrlLow('http://mbs.vero.lt/simple/verify_low.php?
param1=value1&param2=value2');

?>
```

IP addresses

It is possible to check if the data are truly received from the MBS system by checking which IP address sent the data. IP addresses of the MBS server are as follows:

- **213.226.139.33**
- **213.226.139.42**
- **213.226.139.43**

You will see the MBS server address only during the *server-server* requests (e.g. reports of keywords and subscriptions to the external system). However, this method is not applied for *user-server* requests (e.g. WAP redirect requests) as you will see the user IP address rather than the MBS server address.

In case of SOAP and XML POST interfaces, MBS restricts access from certain IP addresses. If you use at least one service via these interfaces, you should render a list of your server IP addresses.

Annex B – Data structures

Country codes

Code	Country
BG	BULGARIA
CH	SWITZERLAND
CZ	CZECH REPUBLIC
DE	GERMANY
EE	ESTONIA
FR	FRANCE
GB	UNITED KINGDOM
HU	HUNGARY
ID	INDONESIA
KZ	KAZAKHSTAN
LT	LITHUANIA
LV	LATVIA
PL	POLAND
RU	RUSSIAN

	FEDERATION
UA	UKRAINE

Schema 12. Country codes in a two-letter format defined in ISO 31661 (Alpha2)

Language codes

Code	Language
BG	Bulgarian
DE	German
EN	English
ET	Estonian
FR	French
HU	Hungarian
ID	Indonesian
IT	Italian
LT	Lithuanian
LV	Latvian
PL	Polish
RM	Raeto-Romance
RU	Russian
UK	Ukrainian

Schema 13Language codes in a format defined in ISO 6391

Currency codes

Code	Currency
BGN	Bulgarian Lev
CHF	Swiss Franc
CZK	Czech Koruna
EEK	Kroon
EUR	Euro
GBP	Pound Sterling
HUF	Forint
IDR	Rupiah
LTL	Lithuanian Litas
LVL	Latvian Lats
PLN	Zloty
USD	US Dollar

Schema 14Currency codes in a format defined in ISO 4217

Operator codes

Code	Operator
Lithuania	
bite_lt	Bitė Lietuva
omnitel_lt	Omnitel Lietuva
tele2_lt	Tele2 Lietuva
Latvia	
bite_lv	Bitė Latvia
lmt_lv	LMT Latvia
tele2_lv	Tele2 Latvia
Estonia	
elisa_ee	Elisa Estonia
emt_ee	EMT Estonia
tele2_ee	Tele2 Estonia
Poland	
era_pl	ERA Poland
orange_pl	Orange Poland
playmobile_pl	Play Mobile Poland
plus_pl	Plus Poland
tele2_pl	Tele2 Poland
unknown_pl	Unknown operator Poland
Indonesia	
bakrie_id	Bakrie Telecom (Esia) Indonesia
excelcom_id	Excelcomindo Pratama (Excelcom) Indonesia
flexi_id	Telkom (Flexi) Indonesia
hti_id	Hutchison CP Telecommunications (HTI) Indonesia
im3_id	Indosat (IM3) Indonesia
m8_id	Mobile-8 Telecom (M8) Indonesia
natrindo_id	Natrindo Telepon Selular (Axis) Indonesia
sampoerna_id	Sampoerna Telekomunikasi (STI) Indonesia
satelindo_id	Indosat (Satelindo) Indonesia
smart_id	Smart Telecom Indonesia
telkomsel_id	Telkomsel Indonesia

Schema 15Operator codes consisting of the abbreviation of the operator and its country code

Transaction state and status

Status	Description
not_finished	Transaction is in progress or not finished
commit	Transaction is completed, the money was successfully charged from the user's account
rollback	Transaction is completed but the money was not charged from the user's account

	because the later error below (e.g. the service could not be provided)
error_money	Errors of the user's account (e.g. low account balance, exceeded account limit, blocked user's account, etc.)
error_operator	Errors in the operator's billing system
error_other	Other errors (e.g. the user refused to pay, misconfigured service, etc.)

Schema 16The main transaction statuses (status)

State	Description
not_finished	
open	Newly created transaction
ac_ready	MBS identified the user, who is the service receiver
ac_done	User agreed to pay for the service
op_wait	Transaction is in progress
commit	
op_done	Transaction is paid for, the service provision is in progress, transaction will be committed or roll backed
commit	Transaction is paid for and committed (the service is provided)
auto_commit	Transaction is paid for and automatically committed (not implemented)
rollback	
rollback	Transaction was paid for but roll backed as the service was not provided
auto_rollback	Transaction was paid for but automatically roll backed as the service was not provided (not implemented)
error_money	
ac_limit	User exceeded its monthly limit in the MBS system
ac_block	User is blocked in the MBS system
ac_flood	MBS blocked too frequent billing of the same user
op_ac_limit	User does not have sufficient money for the payment of the transaction
op_ac_block	User is blocked in the MBS system
op_ac_flood	Operator blocked too frequent billing of the same user
error_operator	
op_cancel_connect	MBS failed to connect to the operator's billing system
op_cancel_data	Operator provided incorrect data
op_cancel_unreach	Operator's system is temporarily unavailable
op_cancel_config	Service is misconfigured in the operator's system
op_cancel_param	Operator provided incorrect payment parameters
op_cancel_unavail	Operator's system is temporarily unable to charge users
op_cancel	Unknown error in the operator's system when charging for the transaction
op_timeout	MBS received no answer from the operator's billing system
error_other	
ac_unknown	MBS was unable to identify the user
ac_cancel	User refused to pay for the service
cancel_config	Service was misconfigured (contact support@vero.lt)
cancel_key	User entered unknown keyword
cancel	Transaction is canceled due to unknown MBS error (contact support@vero.lt)

timeout	Transaction was created but the partner did not use it (not implemented)

Schema 17 Additional transaction statuses (state)

States and statuses of subscribers

Status	Description
active	Active user
suspended	Subscription could not be renewed and was suspended
removed	Removed subscriber

Schema 18The main subscriber's statuses (status)

State	Description
open	Registration of a new user has been just started (short procedure)
active	Active user
active_nosub	The user is active but still not registered in the operator's system (if the service is delayed)
subscribe	The user is not active. It means that WAP registration (initiated by the user) is in progress and the user is redirected to the operator's system to finish the registration. (short procedure)
subscribe_cancel	Unsuccessful user's registration (e.g. error of the operator's system)
subscribe_cancel_account	Unsuccessful user's registration (the user canceled the registration)
subscribe_cancel_limit	Unsuccessful user's registration (the user did not have enough money)
active_bill	User is being charged for the service renewal (short procedure)
suspended	Subscription could not be renewed and was suspended
remove_wait	Subscription is canceled in the operator's system but full removal will be completed after the expiry of the subscription period
remove_op_wait	User requested to cancel her/his subscription, the unsubscription is anticipated from the operator's system
removed	Inactive user (removed from the operator's system)

Schema 19Additional subscriber statuses (state)

User

A user (sometimes called "Account" in the MBS system) is a unique mobile connection account in the MBS system.

- **ACCOUNT_ID** – a unique user ID in the MBS system;
- **PHONE** – a user's telephone number (MSISDN, e.g. 37069923232); displayed not with all operators
- **OPERATOR** – a user's operator (e.g. bite_lt, bite_lv, omnitel_lt, ect.);
- **COUNTRY** – a user's country code (in ISO 3166-1 alpha-2 standard);

- **LANGUAGE** – a user's language (in ISO 639-1 standard).

Services

Service is something that can be sold or provided (e.g. to sell mp3 melody, to provide membership in a club, etc.). In the MBS system it is not possible to buy without a service, therefore, projects of the third parties have to charge users for a service. Key information of any service:

- **ID** – unique service code in the MBS system;
- **TYPE** – service type (WAP payment, subscription payment);
- **NAME** – service name that briefly describes a service to be provided for a user;
- **URL** – the project's address where a service will be provided.

Parameters necessary for every language of service:

- **LANGUAGE** – language you want to show text in;
- **TITLE** – page heading in a language selected;
- **DESCRIPTION** – service short description in a language selected;
- **TEXT** – service question in a language selected.
- **MESSAGE** – text of billing SMS (if billing message is sent) in a language selected.

MBS may provide services of several payments. Below you will find a detailed description of every service type.

WAP services

As the name indicates, this service type is used for one-off purchases while browsing WAP (e.g. to buy a melody, wallpaper, one-off payment for access to the project, etc.). Service type specific parameters:

- **PAID_URL** – address (URL) where a user will be redirected in case of a successful payment for a service;
- **NOT_PAI_URL** – address (URL) where a user will be redirected in case of a refusal to pay for a service;
- **ERROR_URL** – address (URL) where a user will be redirected in case MBS fails to charge a user (e.g. a user has no money in the account, the operator's system does not operate, etc.);
- **BACK_URL** – address (URL) with which a user returns to the project by clicking "Back" button;
- **QUESTION** – this field determines whether a question (Do you agree to pay?) will be displayed before charging a user or not;
- **SECRET_KEY** – Secret word used for security signatures (if not set, partner global secret word will be used)
- **SUBSCRIPTION_SERVICE** – if this WAP service is able to register to subscription, it displays to

what service a subscription should be registered to;

Subscription services

- **PERIOD** – period in hours indicating a time interval when a user should be charged (e.g. 168 so that a user could be charged on a weekly basis);
- **REGISTER_PERIOD** indicates a number of periods for a free of charge subscription renewal (e.g. if REGISTER_PERIOD = 2 and PERIOD = 168, during the first two weeks a user will not be charged);
- **MULTIPLE** indicates whether it is possible to register one user to subscription more than once (currently not available);
- **FORCE_CHANNEL_ID** – a channel used to register users. If it is not set here, a WAP service channel will be used ;
- **NOTIFY_LEVEL** indicates notices on changes in subscription to be sent to a partner (none – none notices will be sent; stats – only notices about subscription facts; all – all events; debug – all events and additional information);
- **NOTIFIER** – a method used to sent notices on events (currently only NotifierPOST is operating. It sends notices using HTTP POST query);
- **NOTIFY_URL** – address (URL) used to send notices on events;
- **SECRET_KEY** – password used to encrypt notices on events sent to a partner (if it is not set here, a main password of a partner will be used);

Transaction

A transaction indicates any billing event (every payments has its own transaction). A transaction has to include the following information:

- **ID** – unique transaction ID;
- **IDENT** – the partner's transaction ID. This parameter is not necessary. It can be used in your application to indicate what exactly you have sold (e.g. a melody's number, name, etc.);
- **STATE** – it shows current transaction state. Possible variants:
- **SERVICE_ID** – a service this transaction belongs to;
- **CHANNEL_ID** – a channel used to sell a service;
- **ACCOUNT_ID** – a unique user ID in the MBS system that received the service;
- **OPERATOR** – the operator whose user was provided with a service (e.g. bite_lt, bite_lv, omnitel_lt, etc.);
- **TARIFF_PRICE** – a service price charged for a transaction (indicated in hundredths of a currency unit, e.g. Lithuanian cents);
- **TARIFF_CURRENCY** – currency used to make a transaction (in ISO 4217 standard);
- **ERROR** – if an error occurs during the transaction, it is specified;
- **LANGUAGE** – a language used to make a transaction (in ISO 639-1 standard);
- **CREATE_DATE** – the creation of a transaction;

- **END_DATE** – the end of transaction (in case of payment error, end or cancellation of a transaction).

Subscriber

- **ACCOUNT** – a user registered to subscription;
- **SUBSCRIBE_SERVICE** – a subscription service;
- **SUBSCRIBE_CHANNEL** – a subscription service channel;
- **STATE** – current state (active membership, terminated or remove membership);
- **SUBSCIBE_PERIOD** – time in hours between a renewal of subscription;
- **LANGUAGE** - Language code (in ISO 639-1 standard), which was used while registering and will be used (by default) when unregistering account from subscription.

Annex C – MBS parameters formation

MBS URL addresses

After the payment (successful or unsuccessful) all addresses (URL) may have dynamic variables:

Name	Alternative	Value
%s	%{service_id}%	Will be changed by service ID
%t	%{transaction_id}%	Will be changed by transaction ID used
%i	%{transaction_ident}%	Will be changed by unique event (transaction) identificator in external system
%r	%{transaction_status}%	Will be changed by payment transaction main status (status). For a list of possible values see 'Transaction Status'
%rs	%{transaction_state}%	Will be changed by payment transaction additional status (state), which accurates reason of failed payment. For a list of possible values see 'Transaction Status'
%p	%{transaction_price}%	Will be changed by paid price (indicated in cents - 1/100 currency)
%cur	%{transaction_currency}%	Will be changed by paid currency (ISO 4217 standard)
%n	%{operator_name}%	Will be changed by a name of user's operator (e.g. bite_lt, omnitel_lt, etc.)
%om	%{account_msisdn}%	Will be changed to user's MSISDN
%a	%{account_id}%	Will be changed by user ID
%o	%{operator_id}%	Will be changed by user's operator ID
%c	%{channel_id}%	Will be changed by channel ID used

Example: the address (URL) set

```
http://wap.mypage.come/after_payment.php?transaction=%t&service=%s&channel=%c&account=%a
```

```
$options = array(
```

```
  array('name' => 'url', 'value' => 'http://wap.manopuslapis.lt/po_apmokejimo.php?
transaction=%t&service=%s&channel=%c&account=%a',
);
```

After the payment a user will be redirected to:

```
http://wap.mypage.com/after_payment.php?
transaction=123455&service=20&channel=20&account=94851
```

Service internationalization

When structuring internationalized phrases for services, dynamic variables may be used in text:

In texts all variables are written between symbols %{ and }%, e.g. %{transaction_id}%

Name	Value	Example
transation_id	Currently running transaction ID	12345
service_id	Used service ID	20
service_name	Used service name	My melodies
channel_id	Used channel ID	10
channel_name	Used channel name	For club members
account_id	Unique user ID	14532
operator_name	Systemic operator's name	bite_lt
operator_title	Operator's name	Bitė Lithuania
language_code	Code of a language used to display a question	LT
language_name	Name of a language used to display a question	Lithuanian
price	Price in the main currency	1
price_not_formed	Price in hundredths of the currency (cents)	100
currency	Currency	LTL
price_formated	Price formatted with currency	1 LTL

Example: when setting for a WAP service TITLE: %{service_name}% TEXT: Do you agree to pay %{price_formated}% ?, a user will see: My melodies Do you agree to pay LTL 1?

Subscription services also have additional phrases:

Name	Value	Example
sub_service_id	Subscription service ID	21
sub_service_period	Payment period in days	3

sub_service_period_hours	Payment period in hours	72
sub_register_period	Free of charge period in days	3
sub_period_price	Price for one period in the main currency unit	1
sub_period_currency	Currency used to charge for one period	LT
sub_period_price_formated	Price formatted for one period	1 LTL

Annex D – Descriptions of Remote Methods

MBSLib – Library in PHP language

You can download the documentation of Vero library here.

SOAP for WAP services

This server is used to create WAP transactions, receive their status, confirm and cancel.

- SOAP server: http://bill.vero.lt/soap/wap1.1
- SOAP server's WSDL: http://bill.vero.lt/soap/wap1.1?wsdl

At the moment project written by JAVA technologies cannot use SOAP methods, because of not valid WSDL

MBSCreateTransaction

This method is used to create a transaction in the MBS system. When a transaction is created, a user has to be redirected to http://bill.vero.lt/prepare?transaction= by adding the transaction number received.

A dynamic address (URL), where a user will be redirected after the billing, may be transferred in the options massive. More about that: Annex C - MBS parameters formation

Request

Parameter	Type	Description	Example
service_id	integer	WAP service ID	1
channel_id	integer	Channel ID	2
language_ident	string	Two-letter language code in ISO 639-1 standard	lt
options	Array of JarrayStruct	Additional parameters, e.g. session ID. If one of the parameters is "url" with value, after the billing a user will be redirected to this address.	Array(array('name'=>'session_id', 'value'=>'d1830eacd56740a920c7099a021f0cc9'))
ident	string	Unique purchase ID in the project (not in MBS). May be empty	My-melodies-123

Response

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	1245634

MBSGetTransaction

This method is used to receive information on the existing transaction.

Request

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	1245634

Response

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	1245634
service_id	integer	Unique WAP service ID	1
channel_id	integer	Unique Channel ID	2
account_id	integer	Unique user ID in the MBS system. If a user is not ascribed to the transaction, the system will return 0.	4495
state	string	Transaction state, in a more detail	op_done

MBSGetTransactionInState

This method is used to receive information on a transaction existing in a certain state.

Request

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	1245634
state	string	Transaction state, in a more detail	op_done

Response

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	1245634
service_id	integer	Unique WAP service ID	1
channel_id	integer	Unique Channel ID	2
account_id	integer	Unique user ID in the MBS system. If a user is not ascribed to the transaction, the system will return 0.	4495
state	string	Transaction state, in a more detail	op_done

MBSCommitTransaction

This method is used to confirm a transaction and money charge.

Request

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	12374

Response

Parameter	Type	Description	Example
success	boolean	Was a transaction successful?	True

MBSRollbackTransaction

This method is used to cancel a transaction. Return of money to a user.

Request

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	12374

Response

Parameter	Type	Description	Example
success	boolean	Was a transaction canceled successfully?	True

SOAP for subscription services

This server is used to manage user subscriptions: register/unregister/check user status.

- SOAP server: http://bill.vero.lt/soap/subscription
- SOAP server's WSDL: http://bill.vero.lt/soap/subscription?wsdl

SubscriptionWapRegister

A user's registration to subscription. If an error is not received after the query, a user has to be redirected to a reference got in the response.

Request

Parameter	Type	Description	Example
wap_service_id	integer	WAP service ID with an enabled subscription service	22
channel_id	integer	Channel ID	1
language_ident	integer	Two-letter language code in ISO 639-1 standard	lt
options	Array of JarraySt ruct	Additional parameters, e.g. session ID. If one of the parameters is "url" with value, after the charge a user will be redirected to this address.	Array(array('name' => 'url', 'value' => 'http://vero.lt'))
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123
valid_till	datetime	Subscription's expiration date	2008-12-12 13:45:32

Response

Parameter	Type	Description	Example
transaction_id	integer	Unique transaction ID	34054
url	string	Address where a user will have to be redirected to be registered to subscription.	http://bill.vero.lt/subscribe?tranzaction=34054

SubscriptionWapUnregister

A user's unregistration from subscription. If an error is not received after the query, a user has to be redirected to a reference got in the response.

Request

Parameter	Type	Description	Example
wap_service_id	integer	WAP service ID registered to subscription	1
language_ident	string	Two-letter language code in ISO 639-1 standard	lt
options	Array of JarrayStruct	Additional parameters	array()
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123

Response

Parameter	Type	Description	Example
url	string	If a user is redirected to this address, he is unregistered from subscription	http://bill.vero.lt/unsubscribe?wap_service_id1=&lang=lt

SubscriptionRegister

Request

Parameter	Type	Description	Example
service_id	integer	Subscription service ID	2
channel_id	integer	Channel ID registered to subscription	1
account_id	integer	User ID which you register to subscription	42314
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123
valid_till	datetime	Subscription's expiration date	2008-12-12 13:45:32

Response

Parameter	Type	Description	Example
url	string	If a user cannot be unregistered remotely, the address where a user should be redirected to unregister will be returned.	http://bill.vero.lt/subscribe?tranzaction=34054

SubscriptionUnregister

Request

Parameter	Type	Description	Example
service_id	integer	Subscription service ID	32
account_id	integer	User ID	4132
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123

Response

Parameter	Type	Description	Example
url	string	If a user cannot be unregistered remotely, the address where a user should be redirected to unregister will be returned.	http://bill.vero.lt/unsubscribe?wap_service_id1=&lang=lt

SubscriptionGet

A user state in the service is checked.

Request

Parameter	Type	Description	Example
wap_service_id	integer	WAP service ID with which a user was registered to subscription	32
service_id	integer	Subscription service ID	2
account_id	integer	User ID which you register to subscription	42314
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123

Response

Parameter	Type	Description	Example
subscriber_id	integer	Unique subscriber (not user) ID	324
account_id	integer	User ID which you register to subscription	42314
service_id	integer	Subscription service ID	2
state	string	Precise subscriber's state	Active, for more information
status	string	Subscriber's state (active, suspended and removed)	removed
channel_id	integer	Channel ID registered to subscription	1
ident	string	Identifier of a project member, not required	vardenis-pavardenis-123
register_date	datetime	Date of registration	2008-01-12 00:00:00
renew_date	datetime	Date of subscription's renewal	2008-02-12 00:00:00
unregister_date	datetime	Date of unregistration	2008-03-12 00:00:00
valid_till	datetime	Subscription's expiration date	2008-03-12 00:00:00

XML POST for WAP services

All queries are made by HTTP POST method to the address (URL) http://bill.vero.lt/remote-xml/wap by putting XML directly into POST.

Parameters of all XML POST methods are analogous to SOAP parameters SOAP for WAP services

Example

```
?php
$xml_document = '<?xml version="1.0" encoding="UTF-8"?>
<MBSCommitTransactionRequest
    xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>1</transaction_id>
</MBSCommitTransactionRequest>';

$handle = curl_init();

curl_setopt($handle, CURLOPT_URL, 'http://bill.vero.lt/remote-xml/wap');
curl_setopt($handle, CURLOPT_RETURNTRANSFER, true);
curl_setopt($handle, CURLOPT_POSTFIELDS, $xml_document);

$response = curl_exec($handle);
if($response){
        echo $response;
}else{
        echo curl_errno($handle).' - '.curl_error($handle);
}
curl_close($handle);
?>
```

MBSCreateTransaction

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSCreateTransactionRequest xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
        <service_id>1</service_id>
        <channel_id>1</channel_id>
        <language_ident>lt</language_ident>
        <options>
            <option name="someName"><![CDATA[http://www.topills.com/a=b&c=d]]></option>
            <option name="someNam1e"><!
[CDATA[http://www.topills.com/a=b&c=d]]></option>
        </options>
        <ident>any ident</ident>
</MBSCreateTransactionRequest>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSCreateTransactionResponse xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
        <transaction_id>1</transaction_id>
        <url><![CDATA[http://wap.site.tld/page?params&other_params]]></url>
        <fault_code>123</fault_code>
        <fault_name>nomessage</fault_name>
        <fault_detail/>
</MBSCreateTransactionResponse>
```

MBSGetTransaction

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSGetTransactionRequest xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>12</transaction_id>
</MBSGetTransactionRequest>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSGetTransactionResponse xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>123</transaction_id>
    <service_id>1</service_id>
    <channel_id>2</channel_id>
    <account_id>3</account_id>
    <state>open</state>
    <fault_code>0</fault_code>
    <fault_name/>
    <fault_detail/>
</MBSGetTransactionResponse>
```

MBSGetTransactionInState

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSGetTransactionInStateRequest  xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>123</transaction_id>
    <state>op_done</state>
</MBSGetTransactionInStateRequest>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSGetTransactionInStateResponse xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>1</transaction_id>
    <service_id>2</service_id>
    <channel_id>3</channel_id>
    <account_id>1</account_id>
    <state>ac_ready</state>
    <fault_code>0</fault_code>
    <fault_name/>
    <fault_detail/>
</MBSGetTransactionInStateResponse>
```

MBSCommitTransaction

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSCommitTransactionRequest
    xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>1</transaction_id>
</MBSCommitTransactionRequest>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSCommitTransactionResponse xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <success>true</success>
    <fault_code>0</fault_code>
    <fault_name/>
    <fault_detail/>
</MBSCommitTransactionResponse>
```

MBSRollbackTransaction

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSRollbackTransactionRequest xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <transaction_id>123</transaction_id>
</MBSRollbackTransactionRequest>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<MBSRollbackTransactionResponse xmlns="http://bill.vero.lt/schema/wap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://bill.vero.lt/schema/wap
http://bill.vero.lt/schema/wap/MBSWapPostXML.xsd">
    <success>true</success>
    <fault_code>0</fault_code>
    <fault_name/>
    <fault_detail/>
</MBSRollbackTransactionResponse>
```

XML POST for subscription services

Not implemented yet.