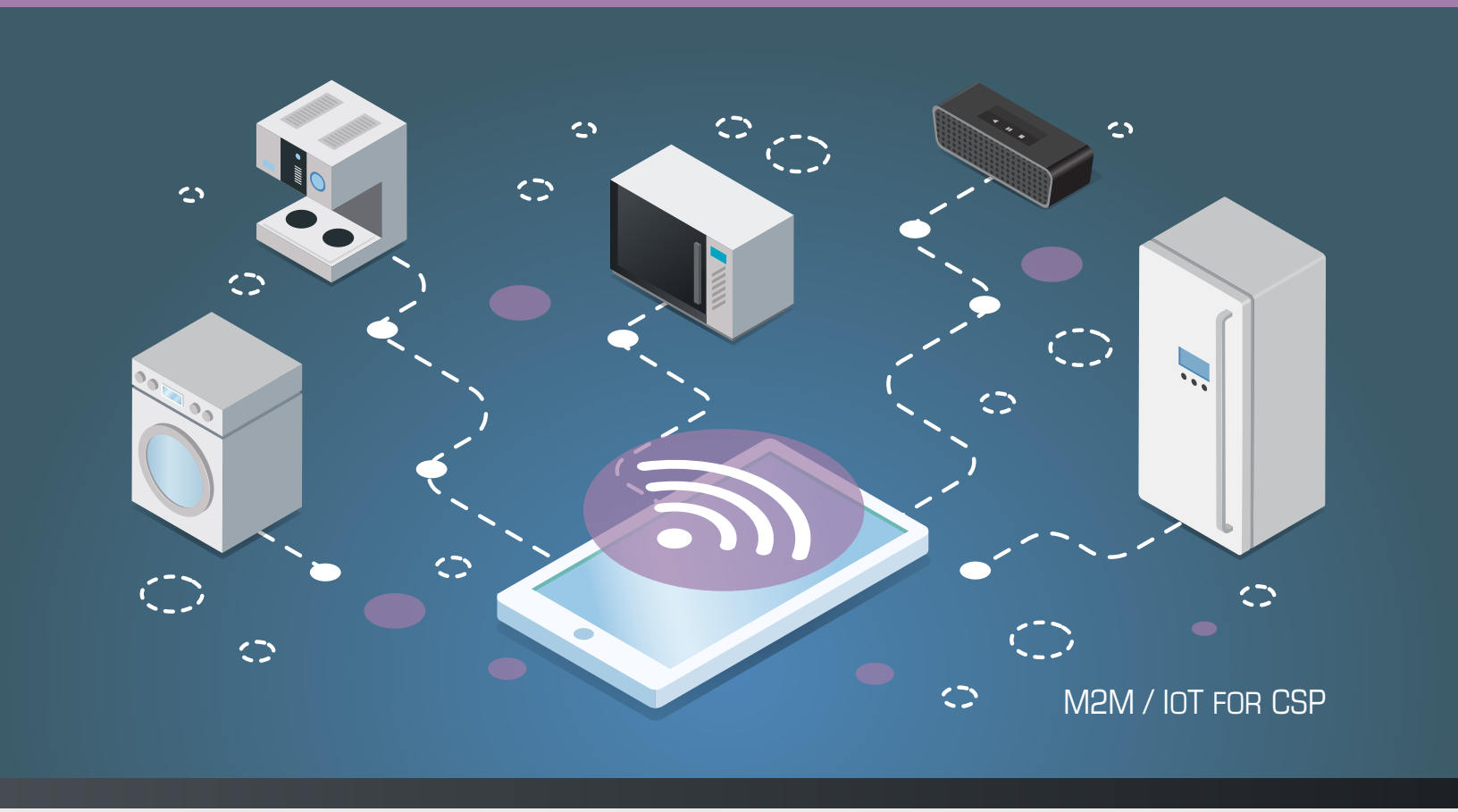
PORTA ONE



Provisioning Application

Reference Guide

75

MAINTENANCE RELEASE

WWW.PORTAONE.COM

Copyright notice & disclaimers

Provisioning Application Reference Guide, February 2019
Maintenance Release 75
V1.75.15

Please address your comments and suggestions to: Sales Department, PortaOne, Inc. Suite #408, 2963 Glen Drive, Coquitlam BC V3B 2P7 Canada.

Table of Contents

Preface

This document provides information for developers who create an external application to provision external systems, such as mobile core network components, IPTV platforms, etc. based on data received from PortaBilling®.

Where to get the latest version of this guide

The hard copy of this guide is updated upon major releases only, and does not always contain the latest material on enhancements that occur in-between minor releases. The online copy of this guide is always up to date, and integrates the latest changes to the product. You can access the latest copy of this guide at: **www.portaone.com/support/documentation/**.

Conventions

This publication uses the following conventions:

- Commands and keywords are given in **boldface**.
- Terminal sessions, console screens, or system file names are displayed in `fixed width font`.

Exclamation mark draws your attention to important actions that must be taken for proper configuration.

NOTE: Notes contain additional information to supplement or accentuate important points in the text.

Timesaver means that you can save time by taking the action described here.

Tips provide information that might help you solve a problem.

Gear points out that this feature must be enabled on the Configuration server.

Trademarks and copyrights

PortaBilling®, PortaSIP® and PortaSwitch® are registered trademarks of PortaOne, Inc.

Overview

When an administrator adds or removes customers in PortaBilling®, or when customer configuration changes (e.g. a credit limit is reached, a product is changed, an invoice is issued or becomes paid, etc.), these changes are recorded as provisioning **events**. The External system provisioning framework (ESPF) monitors these events and sends them via the HTTP/HTTPS protocol to an external application (referred to as the Application in this document).

The Application acts as follows:

- receives provisioning events that contain a unique ID of the entity (e.g. an account) from PortaBilling®,
- retrieves the information about this entity (e.g. account's number, product, service features, etc.) from PortaBilling® via the **PortaBilling API** using the entity's unique ID (e.g. i_account),
- updates the configuration of an external system via this system's API.

Receiving provisional events

The ESPF sends provisioning events in POST requests with the content in the JSON format. Requests are sent in asynchronous mode with the average number of 10 requests per second.

POST requests are sent to the Application's URL. We recommend using the HTTPS transport protocol since it ensures that the communication between PortaBilling® and the application is secure.

Sending parameters such as the Application's URL, HTTP authorization information are defined for the ESPF on the PortaBilling® Configuration server. PortaBilling® administrator provides these data for developers who create an external application.

The POST request contains the following mandatory headers:

- **Date** – This is the originating date and time of the request message in the HTTP date format;
- **Content-Type** – This is the media body type of the request (i.e. application/json).
- **Authorization** – This is authentication information provided by PortaBilling® to authenticate itself with the Application. This header contains the **authentication method** (basic, custom or signature) followed by credentials.

The default method is Basic. This means that PortaBilling® provides base64-encoded user ID and password in the **Authorization** header field.

The body of the POST request contains:

- **event_type** – This is the type of event (created, updated, deleted) that has been applied to a specific entity in PortaBilling® such as:
  - an account,
  - a customer,
  - an invoice,
  - a DID number;
- **variables** – This is the unique ID of the entity that has been modified in PortaBilling®. These are: i_account / i_customer / i_invoice / number (for a DID).

In PortaBilling®, an account record stores information about subscriber's configuration. For compliance with external systems, changes in account's configuration are reflected as events of a **Subscriber** group. A customer record in PortaBilling® stores general information (e.g. invoicing, taxation, etc.) about the owner of acount(s).

Here is an example of the POST request that is sent to the Application:

```
Date: Fri, 11 May 2018 13:28:08 GMT
Authorization: Signature keyId="test",algorithm="hmac-
sha1",signature="b+Y3I1ymQTsuq0h3HNiIl3P3SdE="
Host: 192.168.243.244:5000
Referrer: http://192.168.243.244:5000/
TE: trailers
Content-Length: 83
Content-Type: application/json

{
    "event_type": "Subscriber/Created",
    "variables": {
        "i_account": "1000889"
         }
}
```

To notify the ESPF about the result of provisioning, the Application responds with the **HTTP Status Codes**. Once a response is received, the ESPF acts as follows:

- **200 OK** – The event has been provisioned successfully. The ESPF removes the event from the provisioning queue.
- **4xx Client Error** (e.g. 400 Bad Request) – The event must not be provisioned. The ESPF removes the event from the provisioning queue.

- **Other status code** – An issue appered during provisioning. The ESPF re-sends the event.
- If **no response** is received from the Application during the timeout (5 seconds by default) – the ESPF re-sends the event to the Application.

  Please make sure your application can accept the same provisioning event multiple times.

Please note that the ESPF only interacts with the Application. It considers that the external system (e.g. HSS) has been successfully provisiond once the 200 OK is received. Therefore, in case of provisioning issues between the Application and the external system, make sure that the Application replies with the proper status code.

Retrieiving data from PortaBilling®

When the Application receives a provisioning event, it connects to PortaBilling® via the XML (SOAP) or JSON (REST) API to retrieve the information about the modified entity.

Connection to the XML / JSON API is provided via HTTPS.

To access the XML API, SOAP requests to PortaBilling API must be sent to the following URL:

```
https://portabilling-
web.yourdomain.com:port/soap/service/method
```

To access the REST API, JSON requests to PortaBilling API must be sent to the following URL:

```
https://portabilling-
web.yourdomain.com:port/rest/service/method
```

Replace the **portabilling-web.yourdomain.com** with the actual hostname of the PortaBilling web server.

Replace **'port'** with the required port. The SOAP/JSON interface is available for administrators on port 443.

Replace **'service'** with the API service that contains the required method (e.g. specify the *Account* service to manage account information.)

Replace **'method'** with the required API method (e.g. specify *get_account_info* method in order to get an account record from the database.)

The Content-Type header field used with a HTTPS POST request must have one of the following values:

- application/x-www-form-urlencoded
- multipart/form-data

The body of the POST request must contain the following parameters (in JSON format):

- **auth_info** – The mandatory authentication information such as login-password or login-API token for the admin web interface, or a session ID.
- **params** – A set of method parameters (in JSON format) that depend on a method structure. Note that method parameters and their structures are the same as those in the SOAP.

Here is an example of POST request in JSON format:

```
POST
Request URL:
https://demo.portaone.com:443/rest/Account/get_account_info
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
        "get_included_services": "1"}
    "auth_info": {
        "session_id": "527865ee75368ff2d2c4f4881"
        },
}
```

To access the PortaBilling® web server, the login request must contain a pair: a user login and either the API access token or the user password for the admin web interface. For example:

```
POST
Request URL:https://demo.portaone.com:443/rest/session/login
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
   "params": {
      "password": "exAmple",
      "login": "demo",
      },
}
```

The response returns the session ID value:

```
{"session_id":"527865ee75368ff2d2c4f4881cd2758a"}
```

Please note that we strongly recommend to use the `session_id` for further requests. Otherwise, if you use the login-password or the API access token authentication pairs for every request, new sessions will be created and cause additional load to the database.

For more information about PortaBilling API please refer to the **PortaBilling API** guide.

Sample work flow

Let's have a look on how the Application works.

A service provider provisions subscriber details to their HSS when a mobile user is added to, modified (e.g. a phone number / SIM card / product is changed, etc.) or removed from PortaBilling®.

NOTE: External systems may require different configuration parameters. For example, some HSSs only require SIM card details to activate a SIM card while others require SIM card details and a profile name. That is why the information that the Application requests from PortaBilling® depends on the external system to be provisioned.

In our example we assume that HSS requires SIM card details such as MSISDN, IMSI and a profile name that corresponds to the LTE service name.

Example 1. A mobile account is created in PortaBilling

1. PortaBilling sends the POST request with **Subscriber/Created** event type and the **i_account** to the Application.

```
Date: Fri, 11 May 2018 13:28:08 GMT
Authorization: Signature keyId="test",algorithm="hmac-
sha1",signature="b+Y3I1ymQTsuq0h3HNiIl3P3SdE="
Host: 192.168.243.244:5000
Referrer: http://192.168.243.244:5000/
TE: trailers
Content-Length: 83
Content-Type: application/json

{
    "event_type": "Subscriber/Created",
    "variables": {
        "i_account": "1000889"
    }
}
```

2. The Application receives the request.

3. The Application sends a POST request to PortaBilling® to establish an API session:

```
POST
Request URL:
https://demo.portaone.com:443/rest/session/login
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "password": "exAmple",
        "login": "demo",
        },
}
```

4. The Application receives the response from PortaBilling® with the **session_id.**
5. The Application takes the **i_account** and the **session_id** and calls the following PortaBilling API methods to retriev subscriber details such as service (e.g. LTE) and SIM card details (e.g. MSISDN, IMSI):
    - *Account/get_account_info* to get the list of included services and ensure that that the LTE service is enabled for this subscriber.

```
POST
Request URL:
https://demo.portaone.com:443/rest/Account/get_account_in
fo
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
        "get_included_services": "1"}
    "auth_info": {
        "session_id": "527865ee75368ff2d2c4f4881"
        },
}
```

    - *SIMCard/get_card_list* to get the MSISDN and IMSI.

```
POST
Request URL:
https://demo.portaone.com:443/rest/SIMCard/get_card_list
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
         }
```

```
    "auth_info": {
        "session_id": "527865ee75368ff2d2c4f4881"
        },
}
```

6. Once the subscriber's information is received, the Application interacts with the HSS via the HSS API to add a new subscriber with the following parameters:
    - MSISDN: 12065551122
    - IMSI: 310019901000045
    - Profile name: LTE
7. Once the subscriber is provisioned, the Application replies to PortaBilling® with **200 OK** status code.
8. The ESPF removes the event from the event queue.

Example 2: The existing subscriber has been updated (a new SIM card is assigned)

1. PortaBilling sends the POST request with the **Subscriber/Updated** event type and the **i_account** to the Application.

```
Date: Fri, 21 May 2018 13:28:08 GMT
Authorization: Signature keyId="test",algorithm="hmac-
sha1",signature="b+Y3I1ymQTsuq0h3HNiIl3P3SdE="
Host: 192.168.243.244:5000
Referrer: http://192.168.243.244:5000/
TE: trailers
Content-Length: 83
Content-Type: application/json

{
    "event_type": "Subscriber/Updated",
    "variables": {
        "i_account": "1000889"
    }
}
```

2. The Application receives the request.
3. The Application verifies that the API session is active and reuses the session ID for the request. Otherwise, the application establishes a new API session.
4. The Application uses the **i_account** and the **session_id** to call the following PortaBilling API methods:

- *Account/get_account_info* to get the list of included services, account status and account balance.

```
POST
Request URL:
https://demo.portaone.com:443/rest/Account/get_account_i
nfo
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
        "get_included_services": "1"}
    "auth_info": {
        "session_id": "731865ee75368ff2d2c4f4881"
        },
}
```

- *SIMCard/get_card_list* to get the MSISDN and IMSI.

```
POST
Request URL:
https://demo.portaone.com:443/rest/SIMCard/get_card_list
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
        }
    "auth_info": {
        "session_id": "731865ee75368ff2d2c4f4881"
        },
}
```

5. Upon the response from PortaBilling®, the Application requests the SIM card details from HSS via its API.
6. The Application compares the SIM card parameters received from PortaBilling (MSISDN: 12065551122, IMSI: 310685901111133) with the ones received from the HSS (MSISDN: 12065551122, IMSI: 310685900000045).
7. The Application detects that the IMSI has changed from 310685900000045 to 310685901111133 and notifies the HSS to delete a subscriber with the IMSI: 310685900000045.
8. The Application then instructs the HSS to add a new subscriber with the following parameters:
    - MSISDN: 12065551122
    - IMSI: 310685901111133
    - Profile name: LTE

If the Application detects that the account's status has changed to blocked or suspended, it notifies the HSS to block a SIM card.

If the Application detects that the account has no available funds or has reached the credit limit, it notifies the HSS to act respectively. Note that the actions here depend on the requirements of the HSS.

9. Once the HSS is updated, the Application responds to PortaBilling® with **200 OK** status code.
10. The ESPF removes the event from the event queue.

Example 3: The existing subscriber has been terminated

1. PortaBilling sends the POST request with **Subscriber/Deleted** event type and the **i_account** to the Application

```
Date: Fri, 11 May 2018 13:28:08 GMT
Authorization: Signature keyId="test",algorithm="hmac-
sha1",signature="b+Y3I1ymQTsuq0h3HNiIl3P3SdE="
Host: 192.168.243.244:5000
Referrer: http://192.168.243.244:5000/
TE: trailers
Content-Length: 83
Content-Type: application/json

{
    "event_type": "Subscriber/Deleted",
    "variables": {
        "i_account": "1000889"
    }
}
```

2. The Application verifies that the API session is active and reuses the session ID for the request. Otherwise, the application establishes a new API session.

3. The Application uses the **i_account** and the **session_id** to call the following API methods:
    - *Account/get_account_info* to get the MSISDN (i.e. account ID).

```
POST/HTTP/
Request URL:
https://demo.portaone.com:443/rest/Account/get_account_info
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
         }
```

```
    "auth_info": {
        "session_id": "999865ee75368ff2d2c4f4881"
        },
}
```

- *SIMCard/get_card_list* method to verify that the sim card is no longer assigned to the account.

```
POST/HTTP/
Request URL:
https://demo.portaone.com:443/rest/SIMCard/get_card_list
Host:demo.portaone.com:443
Content-Length: 83
Content-Type: application/x-www-form-urlencoded\r\n

{
    "params": {
        "i_account": "1000889"
         }
    "auth_info": {
        "session_id": "999865ee75368ff2d2c4f4881"
        },
}
```

4. Upon the response from PortaBilling, the Application notifies the HSS to remove / terminate a subscriber with 12065551122 MSISDN.
5. Once the SIM card is removed from the HSS, the Application replies to PortaBilling® with **200 OK** status code.
6. The ESPF removes the event from the event queue.

Event types

Overview

Event types are divided into four groups: subscriber, customer, invoice and DID. Events from each group notify the Application that a corresponding entity has been created, updated or deleted in PortaBilling®. Subscribe the Application for the necessary group of event types:

1. The **Subscriber** group notifies the Application about changes in an account's service configuration and /or billing changes, such as depletion of available funds, block or suspension, etc.

   In PortaBilling®, an account record represents a means via which a user gains access to a service (e.g. a SIM card, a phone line, an IP PBX, etc.). Account records are identified by their unique IDs (i_account). They store an account's service configuration (e.g. product name, service password, status, SIM card, etc.) and billing

configuration (e.g. available funds for debit accounts, credit limit for credit accounts). Some configuration parameters of an account are inherited from a customer.

2. A **Customer** group notifies the Application about changes in a customer's record in PortaBilling®.

   In PortaBilling®, a customer record represents an owner of accounts (e.g. a residential user or a company). Customer records are identified by their unique IDs (i_customer). They store billing information and service configuration that may be inherited by their accounts. Billing information includes the charged amount (balance) and credit limit for postpaid customers or available funds for prepaid customers, invoicing configuration, DID pricing parameters, taxation, etc.

3. An **Invoice** group notifies the Application about invoices that have been generated for or paid by a customer.

4. A **DID** group notifies the Application about DID numbers that have been added to PortaBilling®, used by customers or removed from PortaBilling®.

Group: Subscriber

Subscriber/Created

Description: A new **Account** entity with a unique account ID (e.g. a phone number, username, IP address, etc.) has been added to PortaBilling®.

Subscriber/Updated

Description: The configuration of an existing account has been changed. For example, an account has expired, there are no available funds, a product has been changed for a user, a user has topped up their balance, etc.

Subscriber/Deleted

Description: An account has been permanently terminated in PortaBilling. The user can no longer use the services. The account's status is changed to **Closed** and the account record is removed from the web interface.

Variables	Mandatory	Data type	Nillable	Description
i_account	Yes	integer	No	The unique account ID in PortaBilling®.

Group: Customer

Customer/Created

Description: A new **Customer** entity with a unique customer ID (e.g. a person's or company's name) is added to PortaBilling® when a user subscribes for a service.

Customer/Updated

Description: A customer configuration has been changed in PortaBilling®. For example, the customer's name was changed from Alice Doe to Alice Roe, the customer has exceeded the credit limit, etc.

Customer/Deleted

Description: A customer has been permanently terminated in PortaBilling and can no longer use the services (e.g. the contract has completed, the invoices are not paid, etc.). The customer's status is first changed to **Closed** and then the customer record is removed from the web interface.

Variables	Mandatory	Data type	Nillable	Description
i_customer	Yes	integer	No	The unique ID of a customer in PortaBilling®.

Group: Invoice

Invoice/Created

Description: An invoice with a unique ID has been generated for a customer in PortaBilling®.

Invoice/Updated

Description: An invoice with a unique ID has been paid fully or partially or refunded in PortaBilling®.

Variables	Mandatory	Data type	Nillable	Description
i_customer	Yes	integer	No	A customer's unique ID in PortaBilling®.
i_invoice	Yes	integer	No	An invoice's unique ID in PortaBilling®.

Group: DID

DID/Created

Description: A **DID number** has been uploaded to PortaBilling®.

DID/Updated

Description: A **DID number** has been updated as follows: released to the DID pool, assigned to / removed from a customer or an account, moved to another installation or removed from the DID inventory in PortaBilling®.

DID/Deleted

Description: A **DID number** has been removed from the DID inventory in PortaBilling®.

Variables	Mandatory	Data type	Nillable	Description
number	Yes	string	No	The DID number.

Commonly used PortaBilling® API methods

Since external systems (e.g. mobile core network components, IPTV platforms, etc.) differ, they need different data from PortaBilling® and they require specific actions to be taken.

Here is the list of most commonly used API methods that will help you receive necessary information from PortaBilling®:

- **Account/get_account_info** – Use this method to receive subscriber information, such as account ID (e.g. DID, MSISDN, IPv4 address, login), service password, the unique ID of the product (i.e. i_product) and name, activation and expiration dates, available funds, status (e.g. active / blocked / terminated), service policy (e.g. the unique ID of the access policy) and service features that are enabled for this subscriber, etc.
- **Account/get_custom_fields_values** – Use this method to receive custom information assigned to an account (i.e. db_value).
- **Account/get_service_features_list** – Use this method to retrieve the list of service features that are available for an account.
- **Product/get_product_info** – Use this method to receive product information such as unique IDs of included services (i.e. i_service), subscription (i_subscription) and volume discount plan (i.e. i_vd_plan), etc.

- **Service/get_service_info** – Use this method to receive service details such as service name, rating base, etc.
- **SIMCard/get_card_list** – Use this method to receive SIM card details such as MSISDN, IMSI and the unique ID of the SIM card (i_sim_card), etc.
- **AccessPolicy/get_access_policy_info** – Use this method to receive Internet access policy details such as the unique ID of a service policy (i_service_policy), hotlining parameters, etc.
- **ServicePolicy/get_service_policy_info** – Use this method to receive service policy details such as specific service attributes.
- **DID/get_number_info** – Use this method to receive DID details such as cost and revenue, owner, batch, etc.
- **Invoice/get_invoice_info** – Use this method to receive invoice details such as the status of the invoice (e.g. paid / partially paid / unpaid), the amount already paid by the customer, the amount that must be paid by the customer and the date when the invoice was generated, etc.
- **Customer/get_customer_info** – Use this method to receive customer record details such as balance, billing status (e.g. active / blocked / suspended), billing period, personal information (e.g. salutation, name, address, etc.), IP Centrex configuration, etc.

Please refer to the **PortaBilling® API** guide for a more detailed description of API methods and their structures.

APPENDIX A. Authentication methods

PortaBilling® ESPF supports three HTTP authentication schemes:

- **Basic** – authenticate with user ID and password.

  Credentials string is constructed by joining the username and the password with a single colon (":"). The result is then base64 encoded.

  For example, let's say that user ID is *username* and the password is *secret* (i.e. username:secret). The authorization header then looks as follows:

  ```
  Authorization: Basic dXNlcm5hbWU6c2VjcmV0
  ```

  The Application receives the request, decodes the base64-encoded user ID and password. It compares the decoded credentials username:secret with the ones that are stored in the Application's database. If they match, the authorization is passed.

NOTE: Basic authentication is usually done by the web server that runs the application.

- **Custom** – authenticate with custom type and credentials;

  The **Authorization header** contains the name of a custom authentication scheme and credentials as it is configured in PortaBilling®.

  For example, let's say that custom type is *Plain* and the password is *passexample*. The authorization header then looks as follows:

  ```
  Authorization: Plain passexample
  ```

  The Application receives the request, compares the authentication schema and credentials with the ones that are stored in the Application's database. If they match, the authorization is passed.

- **Signature** – authenticate by signature key and key ID.

  Credentials sting is constructed of the originating date (the value from the Date header), signature key and signature key ID.

  The **Date** header is used to form a **signing string**. This signing string is then signed with the signature **key** to make a **signature**. The HMAC-SHA1 **algorithm** (Hash-based Message Authentication Code using the SHA1 hash function) is used to sign the signing string with the key.

  For example, let's say that the signature key is *signature,* the key ID is *test* and the Date header is *Thu, 12 Apr 2018 15:24:00 GMT*. The authorization header then looks as follows:

  ```
  Authorization: Signature
  keyId="test",algorithm="hmac-
  sha1",signature="+IkiEg9UkyA+gh+pcI64iti
  ```

  The Application receives the request, takes the value from the Date header field and verifies the value from the key ID field. The Application then takes the key value from the Application's database and runs the algorithm to calculate the signature. The Application compares the obtained signature value with the one that is received in the Authorization header. If they match, the authorization is passed.

APPENDIX B. Sample Application to process provisioning events

This is the example of the Application (Perl module) that provisions SIM card details to the HSS. The Application is subscribed to process event types of the Subcrtiber group.

```
#!/usr/bin/perl
# Example Web Service to process events from EventSender
handler
#
# run:
#   PORTA_BILLING_API=10.0.3.6 \
#   PORTA_BILLING_API_USER=api-login \
#   PORTA_BILLING_API_PASSWORD=api-password \
#   RESULT_FILE=/tmp/hss.log \
#   SERVICE_LOGIN=events \
#   SERVICE_PASSWORD=topsecret \
#   plackup --host 127.0.0.1 --port 9090 perl_example.psgi

use strict;
use warnings;
use Const::Fast;
use Cpanel::JSON::XS qw(decode_json encode_json);
use English qw(-no_match_vars);
use HTTP::Status qw(:constants);
use HTTP::Tiny;
use IO::File;
use MIME::Base64 qw(encode_base64);
use Plack::Request;
use POSIX qw(strftime);

const my $RESULT_FILE => ( $ENV{RESULT_FILE} //
'/tmp/hss.log' );

# basic authorization
my $user             = $ENV{SERVICE_LOGIN} // 'events';
my $password         = $ENV{SERVICE_PASSWORD} //
'topsecret';
my $base_auth_string = 'Basic ' . encode_base64( $user . ':'
. $password, '' );

# PortaBilling API server
my $PB_API_HOST     = $ENV{PORTA_BILLING_API} // '10.0.0.1';
my $PB_API_USER     = $ENV{PORTA_BILLING_API_USER} // '';
my $PB_API_PASSWORD = $ENV{PORTA_BILLING_API_PASSWORD} //
'';

# reuse PB API session
my $SESSION_EXPIRATION = $ENV{SESSION_EXPIRATION} // 60;
my ( $session, $session_last_usage );

my $http = HTTP::Tiny->new( verify_SSL => 0, timeout => 5 );

# error logging
sub log_error {
    my $message = shift;
    print STDERR '[ERROR] ', $message, "\n";
    return;
```

```
}

sub log_debug {
    my $message = shift;
    print STDERR '[DEBUG] ', $message, "\n";
    return;
}

# Perform HTTP/REST request to PortaBilling API
sub get_api_result {
    my ( $method, $session_id, $params ) = @_;

    log_debug( sprintf "API: POST https://%s/rest/%s %s",
$PB_API_HOST, $method, encode_json($params) );

    my $response = $http->post_form(
        'https://' . $PB_API_HOST . '/rest/' . $method, {
            auth_info => encode_json( $session_id ? {
session_id => $session_id } : { login => $PB_API_USER,
password => $PB_API_PASSWORD } ),
            params => encode_json($params),
        }
    );
    if ( !$response->{success} ) {
        log_error( sprintf 'PB API %s failed, error %s %s',
$method, $response->{status}, $response->{reason} );
        return undef;
    }

    # debug, if required:
    #print STDERR 'PB API ', $method, ' response: ',
$response->{content}, "\n";

    my $data = eval { decode_json( $response->{content} ) };
    if ( $EVAL_ERROR || !$data ) {
        # no content or malformed JSON
        log_error( sprintf 'Failed to parse reply content:
%s, error %s', $response->{content} // '', $EVAL_ERROR );
        return undef;
    }

    return $data;
} ## end sub get_api_result

# Login to PortaBilling API
sub api_login {
    my ( $api_login, $api_password ) = @_;

    if ( $session_last_usage && $session_last_usage +
$SESSION_EXPIRATION > time() ) {
        # session active
        log_debug( sprintf 'Reusing session %s', $session );
        return $session;
    }

    my $data = get_api_result( 'Session/login', undef, {
login => $api_login, password => $api_password } );
    return undef if ( !$data );

    $session            = $data->{session_id};
    $session_last_usage = time();
    log_debug( sprintf 'Created session %s', $session );
```

```
    return $session;
}

# Get Account information
sub api_get_account_info {
    my ( $session_id, $i_account ) = @_;

    my $data = get_api_result( 'Account/get_account_info',
$session_id, { i_account => $i_account } );
    return undef if ( !$data );
    $session_last_usage = time();
    return $data->{account_info};
}

# Get list of SIM Cards assigned to Account
sub api_get_sim_cards {
    my ( $session_id, $i_account ) = @_;

    my $data = get_api_result( 'SIMCard/get_card_list',
$session_id, { i_account => $i_account } );
    return undef if ( !$data );
    $session_last_usage = time();
    return $data->{card_list};
}

# Here we perform actual provisioning of collected data to
external HSS
# As an example, we just write information to local file
# row format: action,account-id,balance,status,IMSI,datetime
#  where
#   action - string, one of 'Created', 'Updated', 'Deleted'
#   account-id - string, ID of account
#   balance - number, account's balance
#   status - string, account's status
#   IMSI - string, SIM card IMSI (optional)
#   datetime - string, datetime in YYYY-MM-DD hh:mm:ss
format
sub provision_external_system {
    my $h = shift;

    my $status   = 0;
    my $account  = $h->{account};
    my $sim_list = $h->{sim_cards} // [];
    my $datetime = strftime( '%Y-%m-%d %H:%M:%S',
localtime() );

    my $fh = IO::File->new( $RESULT_FILE, 'a' );
    if ( !defined $fh ) {
        log_error( sprintf( 'Failed to open file %s, error
%s', $RESULT_FILE, $OS_ERROR ) );
        return $status;
    }

    if ( scalar( @{$sim_list} ) == 0 ) {
        # Account without SIM cards
        if ( !printf $fh "%s,%s,%.5f,%s,,%s\n", $h-
>{action}, $account->{id}, $account->{balance}, ( $account-
>{status} || 'open' ), $datetime ) {
            log_error( sprintf( 'Failed to write file %s,
error %s', $RESULT_FILE, $OS_ERROR ) );
            $status = 0;
        }
        $status = 1;
```

```
    }
    else {
        foreach my $sim ( @{$sim_list} ) {
            if (
                !printf $fh "%s,%s,%.5f,%s,%s,%s\n", $h-
>{action}, $account->{id}, $account->{balance}, ( $account-
>{status} || 'open' ), $sim->{imsi},
                $datetime
            ) {
                log_error( sprintf( 'Failed to write file
%s, error %s', $RESULT_FILE, $OS_ERROR ) );
                $status = 0;
                last;
            }
            $status = 1;
        }
    }

    if ( !$fh->close ) {
        log_error( sprintf( 'Failed to close file %s, error
%s', $RESULT_FILE, $OS_ERROR ) );
        $status = 0;
    }

    log_debug( sprintf 'Provisioning status: %s', ( $status
? 'OK' : 'FAILURE' ) );

    return $status;
} ## end sub provision_external_system

# check requirements for incoming request
sub validate_request {
    my $req = shift;

    # HTTP method
    if ( $req->method ne 'POST' ) {
        log_error('Only POST method allowed');
        return HTTP_METHOD_NOT_ALLOWED;
    }

    # Basic Authorization
    my $auth_value = $req->header('Authorization') || '';
    if ( $auth_value ne $base_auth_string ) {
        log_error('Auth failed');
        return HTTP_UNAUTHORIZED;
    }

    # require Content-Type: application/json
    if ( $req->content_type ne 'application/json' ) {
        log_error( sprintf 'Content-Type %s, expected
application/json', $req->content_type );
        return HTTP_UNSUPPORTED_MEDIA_TYPE;
    }

    return 0;
} ## end sub validate_request

sub process_request {
    my $req = shift;

    my $code = validate_request($req);
    return $code if ( $code > 0 );
```

```
    # parse request
    my $event_content = $req->content;
    my $event = eval { decode_json($event_content) };
    if ( $EVAL_ERROR || !$event ) {
        # received malformed JSON data: 400 Bad Request
        return HTTP_BAD_REQUEST;
    }

    log_debug(
        sprintf 'Received event: %s Variables: %s', $event-
>{event_type},
        join( ' ', map { $_ . '=' . $event->{variables}-
>{$_} } ( sort keys %{ $event->{variables} } ) )
    );

    # Subscriber/Created
    # Subscriber/Updated
    # Subscriber/Deleted
    #   variables: i_account

    my ( $object, $action ) = split( /\//, $event-
>{event_type}, 2 );

    if ( $object ne 'Subscriber' ) {
        # ignore
        return HTTP_OK;
    }

    my $i_account = $event->{variables}->{i_account};
    if ( !$i_account ) {
        # mandatory variable missing: 400 Bad Request
        return HTTP_BAD_REQUEST;
    }

    my $api_session = api_login( $PB_API_USER,
$PB_API_PASSWORD );
    if ( !$api_session ) {
        log_error('PB API login failed');
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    my $account = api_get_account_info( $api_session,
$i_account );
    if ( !$account ) {
        log_error('Account not found');
        return HTTP_OK;
    }

    my $sim_card_list = api_get_sim_cards( $api_session,
$i_account );
    if ( !$sim_card_list ) {
        log_error('Failed to get SIM Cards for Account');
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    if (
        !provision_external_system( {
                action    => $action,
                account   => $account,
                sim_cards => $sim_card_list,
            }
        )
    ) {
```

```
        # TODO add required error processing (alerts,
retries, etc)
        return HTTP_INTERNAL_SERVER_ERROR;
    }

    return HTTP_OK;
} ## end sub process_request

# PSGI application
my $app = sub {
    my $env = shift;
    my $req = Plack::Request->new($env);

    my $code = process_request($req);

    return $req->new_response($code)->finalize;
};

log_debug('Started');

return $app;
```