Maintenance Release 90

PortaBilling

Provisioning Application Reference Guide

Copyright Notice & Disclaimers

Provisioning Application Reference Guide, February, 2021

Maintenance Release 90

V1.90.01

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 introduced between major 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.

 The **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 performing the action described here.

 **Archivist** explains how the feature worked in previous releases.

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

 **Tips** provide information that might help you solve a problem.

Trademarks and copyrights

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

What is new in maintenance release 90?

Added:

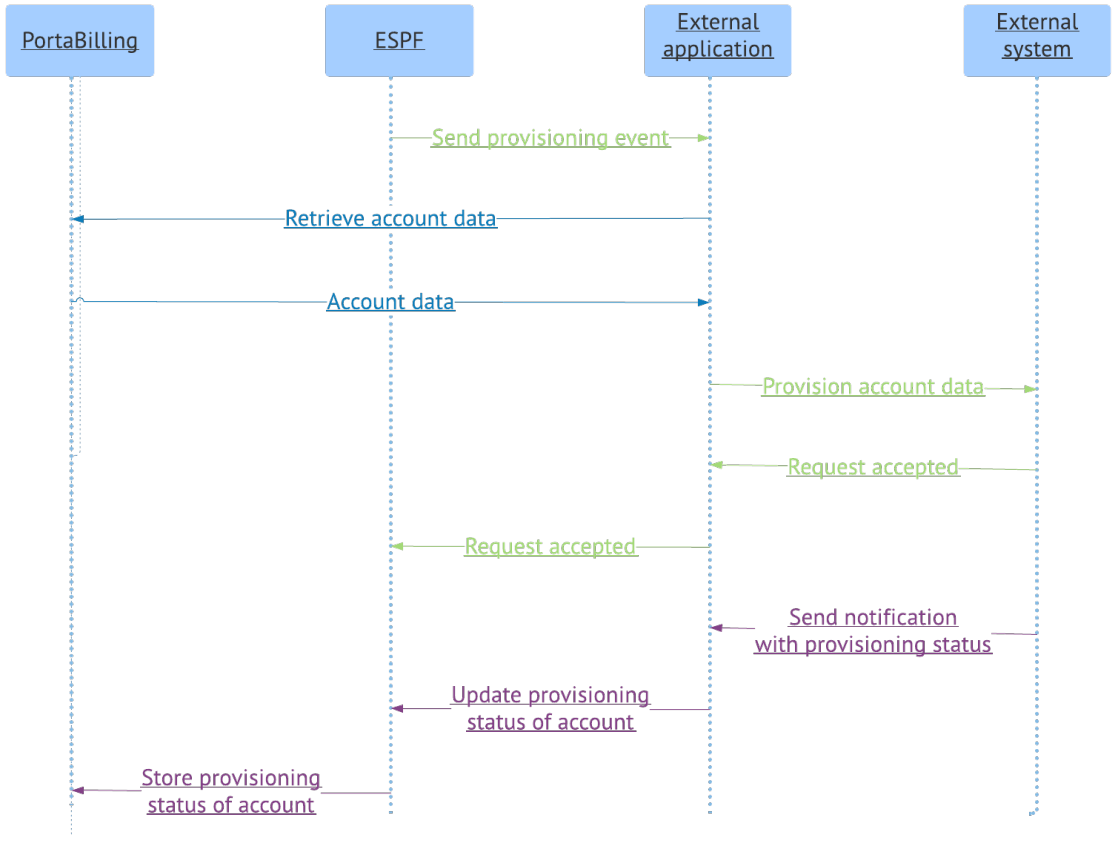
- The **Scheduling an event via ESPF API** chapter.

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,
- updates the account's provisioning status in PortaBilling® via the ESPF 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).
- **i_event** – This is the unique identifier of the event. The Application uses this ID to recognize whether it is a new request or a repeat request and adjust the provisioning flow.
  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 account(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,
        "i_event":  7615
     }
}
```

The Application responds with the **HTTP Status Codes**. Once a response is received, the ESPF acts as follows:

- **200 OK** – The event has been received. 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 appeared during provisioning. The ESPF re-sends the event.
- If **no response** is received from the Application during the timeout (300 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.

Your external system can process provisioning requests synchronously and asynchronously. When processing data synchronously, a system processes a request immediately upon receipt and sends the provisioning results to the Application. Asynchronous data processing means that a system accepts a request and sends notifications with the provisioning status to your Application (e.g. an hour later).

Regardless of the data processing mode, once the Application receives a response from the external system, it can store the provisioning status for an account in PortaBilling®. The Application sends the API request with the account's provisioning status to the ESPF.

Please note that the ESPF only interacts with the Application. It considers that the external system (e.g. HSS) has been successfully provisioned 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.

Retrieving 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 with these parameters (in JSON format):

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

```
POST /rest/Account/get_account_info HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 139

auth_info=%7B%22session_id%22%3A%22527865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%2C%22get_included_services%22%3A1%7D
```

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. Here is an example of the POST request with these parameters:

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

```
POST /rest/Session/login HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 70

params=%7B%22login%22%3A%22demo%22%2C%22password%22%3A%22exAmple%22%7D
```

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.

Sending provisioning status for an account via ESPF API

When the Application receives the response from the external system, it connects to the ESPF via the API and sends the HTTPS POST request to update the provisioning status for an account in PortaBilling® database.

The ESPF API is available under this URL:

https://<portabilling-web.yourdomain.com>/espf/v1/account/i/<i_account>/status

where:

- `portabilling-web.yourdomain.com` is the hostname of your PortaBilling® web server;
- `v1` is the provisioning API version;
- `i_account` is the unique ID of the PortaBilling® account; and
- `status` is the account's provisioning status.

If you operate with account ID rather than i_account, then the URL changes to
https://<portabilling-web.yourdomain.com>/espf/v1/account/<id>/status

Authorization is done using the API credentials of the PortaBilling® administrative user via one of the available **authentication methods**.

The Content-Type header field used with a HTTPS POST request must be of the `application/x-www-form-urlencoded` type.

The body of the request must include these parameters:

Attribute	Mandatory	Type	Description
status	Y	string	The account's provisioning status. Possible values: • OK – the provisioning is successful; • IN_PROGRESS – the provisioning is in progress; • RETRY – a retry is initiated after an initial failure; • FAILED – the provisioning failed.
group	Y	string	A service group organizes the provisioning requests per the external system (e.g. the IPTV group is used to identify provisioning requests to an IPTV platform). When the Application sends the API requests with the account's provisioning statuses, they are distinguished and stored under the corresponding service group name.

			You can use any group name to store the account's provisioning status. First, register this group in PortaBilling® and then pass its name to the ESPF in the API request. Please refer to the **Manage service groups** section for details. These groups are pre-defined in PortaBilling®: • VoIP ; • HSS; • PCRF; • IPTV; • WiMAX; • Netaccess; • NumberPortability; and • LawfulInterception.
error	N	string (max 4000 chars)	The message that describes the failed result of setting an account's provisioning status. This attribute is ignored for the OK status value.

Here is an example of the POST request:

```
POST /espf/v1/account/i/189/status HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 19

status=OK&group=HSS
```

The response from the ESPF contains the **HTTP Status Code**.

Manage service groups

The ESPF API contains methods to retrieve the list of service groups available to store account provisioning status and to add and delete custom service groups.

Use these HTTP methods to manage service groups:

- GET – to get the service group list;
- POST – to add a new custom service group;
- DELETE and POST – to delete a custom service group. DELETE is the recommended method.

get_group_list

Use this method to retrieve the list of service groups defiend for ESPF from the database.

The URL to send requests to is:
`https://<portabilling-web.yourdomain.com>/espf/v1/groups`

The response includes these attributes in JSON format:

Attribute	Type	Description
group_list	Array of **GroupList structure**	The list of service groups registered in PortaBilling®

GroupList structure

Attribute	Type	Description
name	string	A service group name
i_group	int	The unique ID of a service group
is_custom	string	Indicates whether a service group is the predefined or a custom one. Possible values: • true – the group is custom; • false – the group is pre-defined

Request example:

```
GET /espf/v1/groups HTTP/1.1
Authorization: Basic dXNlciBwQHNzdzByZA==
User-Agent: curl/7.29.0
Host: 192.168.248.32
Accept: */*
Content-Type: application/json
```

Response example:

```
{
   "group_list":[
      {
         "i_group":"1",
         "name":"VoIP",
         "is_custom":false
      },
      {
         "i_group":"2",
         "name":"HSS",
         "is_custom":false
      },
      {
         "i_group":"3",
         "name":"PCRF",
         "is_custom":false
      },
      ......
      {
         "i_group":"1001",
         "name":"IOT",
         "is_custom":true
      }
   ]
}
```

create_group

Use this method to add a custom service group to the database.

The URL to send requests to is:

`https://<portabilling-web.yourdomain.com>/espf/v1/group/create`

The body of the request must include the service group name.

The response contains the information about a new group in JSON format:

Attribute	Type	Description
name	string (max 64 chars)	A service group name
i_group	int	The unique ID of the service group

Request example:

```
POST /espf/v1/group/create HTTP/1.1
Authorization: Basic dXNlciBwQHNzdzByZA==
Host:demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 8

name=DPI
```

Response example:

```
{
   "group":{
      "i_group":"1002",
      "name":"DPI"
   }
}
```

delete_group

Use this method to remove a service group from PortaBilling®. You can delete only custom groups. Upon method execution the group specified is marked as obsolete in the database.

The URL to send requests to is:
`https://<portabilling-web.yourdomain.com>/espf/v1/group/<name>`

where:

`<name>` is the name of a custom service group.

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"
        "i_event": "5"
    }
}
```

2. The Application receives the request.
3. The Application sends a POST request to PortaBilling® to establish an API session.
   Used parameters:
   `params={"login":"demo","password":"exAmple"}`

```
POST /rest/Session/login HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 70

params=%7B%22login%22%3A%22demo%22%2C%22password%22%3A%22exAmple%22%7D
```

4. The Application receives the response from PortaBilling® with the **session_id.**
5. The Application takes the **i_account** and the **session_id** values and calls PortaBilling API to retrieve subscriber details such as service (e.g. LTE) and SIM card details (e.g. MSISDN, IMSI). The API methods are:
   - ***Account/get_account_info*** to get the list of included services and ensure that the LTE service is enabled for this subscriber.
     Used parameters:
     `auth_info={"session_id":"527865ee75368ff2d2c4f4881"}`
     `params={"i_account":1000889,"get_included_services":1}`

```
POST /rest/Account/get_account_info HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 139

auth_info=%7B%22session_id%22%3A%22527865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%2C%22get_included_services%22%3A1%7D
```

   - ***SIMCard/get_card_list*** to get the MSISDN and IMSI.
     Used parameters:
     `auth_info={"session_id":"527865ee75368ff2d2c4f4881"}`
     `params={"i_account":1000889}`

```
POST /rest/SIMCard/get_card_list HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 105

auth_info=%7B%22session_id%22%3A%22527865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%7D
```

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

Once the subscriber is provisioned, the Application replies to PortaBilling® with **200 OK** status code.

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"
        "i_event": "6"
    }
}
```

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.

Used parameters:
auth_info={"session_id":"527865ee75368ff2d2c4f4881"}
params={"i_account":1000889,"get_included_services":1}

```
POST /rest/Account/get_account_info HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 139

auth_info=%7B%22session_id%22%3A%22527865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%2C%22get_included_services%22%3A1%7D
```

- ***SIMCard/get_card_list*** to get the MSISDN and IMSI.
  Used parameters:
  auth_info={"session_id":"527865ee75368ff2d2c4f4881"}
  params={"i_account":1000889}

```
POST /rest/SIMCard/get_card_list HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 105

auth_info=%7B%22session_id%22%3A%22527865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%7D
```

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"
        "i_event": "8"
    }
}
```

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).
     Used parameters:
     `auth_info={"session_id":"999865ee75368ff2d2c4f4881"}`
     `params={"i_account":1000889}`

```
POST /rest/Account/get_account_info HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 105

auth_info=%7B%22session_id%22%3A%22999865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%7D
```

- ***SIMCard/get_card_list*** method to verify that the sim card is no longer assigned to the account.
  Used parameters:
  `auth_info={"session_id":"999865ee75368ff2d2c4f4881"}`
  `params={"i_account":1000889}`

```
POST /rest/SIMCard/get_card_list HTTP/1.1
Host: demo.portaone.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 105

auth_info=%7B%22session_id%22%3A%22999865ee75368ff2d2c4f4881%22%7D&params=%
7B%22i_account%22%3A1000889%7D
```

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.

The variables set is the same for all event types of this group:

Variables	Mandatory	Type	Nillable	Description
i_account	Yes	int	No	The unique account ID in PortaBilling®.
i_event	Yes	int	No	This is the unique identifier of the event. The Application uses this ID to recognize whether it is a new request or a retry and to adjust the provisioning flow. For example, in the case of a retry, the Application verifies the request's status. If the request is processed, it responds with 200OK; otherwise, it treats the request as new and processes it.

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.

The variables set is the same for all event types of this group:

Variables	Mandatory	Type	Nillable	Description
i_customer	Yes	int	No	The unique ID of a customer in PortaBilling®.
i_event	Yes	int	No	This is the unique identifier of the event. The Application uses this ID to recognize whether it is a new request or a retry and to adjust the provisioning flow. For example, in the case of a retry, the Application verifies the request's status. If the request is processed, it responds with 200OK; otherwise, it treats the request as new and processes it.

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®.

The variables set is the same for all event types of this group:

Variables	Mandatory	Type	Nillable	Description
i_customer	Yes	int	No	A customer's unique ID in PortaBilling®.
i_invoice	Yes	int	No	An invoice's unique ID in PortaBilling®.
i_event	Yes	int	No	This is the unique identifier of the event. The Application uses this ID to recognize whether it is a new request or a retry and to adjust the provisioning flow. For example, in the case of a retry, the Application verifies the request's status. If the request is processed, it responds with 200OK; otherwise, it treats the request as new and processes it.

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®.

The variables set is the same for all event types of this group:

Variables	Mandatory	Type	Nillable	Description
number	Yes	string	No	The DID number.
i_event	Yes	int	No	This is the unique identifier of the event. The Application uses this ID to recognize whether it is a new request or a retry and to adjust the provisioning flow. For example, in the case of a retry, the Application verifies the request's status. If the request is processed, it responds with 200OK; otherwise, it treats the request as new and processes it.

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.

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

Scheduling an event via ESPF API

Using the ESPF API, you can add events to the provisioning queue and schedule the time when they must be processed. You can use either standard system events or custom events created for specific tasks. Refer to the **External System Interfaces Guide** to see the list of available provisioning event types.

The ESPF API is available at this URL:
`https://<portabilling-web.yourdomain.com>/espf/v1/`

where:

- `portabilling-web.yourdomain.com` is the hostname of your PortaBilling® web server;
- `v1` is the provisioning API version.

The authorization is done using the PortaBilling® administrative user's API credentials.

Usage scenario

A CSP provides customers with Internet service in roaming with a monthly quota. When the quota for an account is reached, the service is blocked till the end of the month. The ESPF sends the **Account/Service/QuotaExceeded** event to the Application, the Application queries the account and service details via PortaBilling® API and then interacts with the HSS via the HSS API to block this service for the account.

At the beginning of the next month, the quota must be made available again, if the account has sufficient funds. PortaBilling® allocates the quota after receiving the first authorization request in the new month (when a user tries to access the service). To receive this request, the service must be unblocked in the mobile core.

To unblock the service in the mobile core at the beginning of the new month, the Application can schedule an event with the specified start time via ESPF API.

Thus, when the quota is reached, e.g., on February 20, the Application receives a notification about this event from PortaBilling® and notifies the HSS to block the service for the account. At the same time, the Application calls the ESPF API to add a custom event "the new month has started" scheduled for midnight of March 1, 2021. At this scheduled time, the Application receives a notification about this event from the ESPF. This means that the new month has started and now the Application must check whether the account has sufficient funds and unblock the service in the mobile core.

To configure this usage scenario, perform the following steps.

Make sure that the ESPF is configured to provision events to the Application. The **EventSender** handler must be configured on the Configuration server and enabled. Refer to the **ESPF configuration** handbook for details.

Note that to configure the ESPF components, the **evctl.pl** script is used. To run the script, connect to the server (where the ESPF provisioning is enabled) via SSH and run the following commands.

1. Create a custom event via the command-line interface.

To create a custom event, execute the following command:

```
/home/provisioning-framework/utils/evctl.pl type create <custom event name>
```

Note that the name of the event must start with “Custom/” (e.g., Custom/NextMonth).

For example:

```
/home/provisioning-framework/utils/evctl.pl type create Custom/NextMonth
Created event type #10001 Custom/NextMonth
```

2. Subscribe the EventSender to the created custom event (e.g., Custom/NextMonth). To do this, run the following command:

   ```
   /home/provisioning-framework/utils/evctl.pl handler subscribe EventSender
   <custom event name>
   ```

   ```
   /home/provisioning-framework/utils/evctl.pl handler subscribe EventSender
   Custom/NextMonth
   Handler EventSender is subscribed to Custom/NextMonth
   ```

3. To send notifications to the Application when the quota is reached, enable the **Account/Service/QuotaExceeded** event and subscribe the EventSender handler to it.

   To do this, run the following commands:

   ```
   /home/provisioning-framework/utils/evctl.pl type enable <event name>
   ```

   ```
   /home/provisioning-framework/utils/evctl.pl type enable
   Account/Service/QuotaExceeded
   Triggers for table Account_VD_Counters AFTER INSERT were recreated
   Triggers for table Account_VD_Counters AFTER UPDATE were recreated
   Event Account/Service/QuotaExceeded was enabled
   ```

   ```
   /home/provisioning-framework/utils/evctl.pl handler subscribe EventSender
   <event name>
   ```

   ```
   /home/provisioning-framework/utils/evctl.pl handler subscribe EventSender
   Account/Service/QuotaExceeded
   Handler EventSender is subscribed to Account/Service/QuotaExceeded
   ```

   Note that the Application may require additional subscriptions (for example, Account/New to create an account in the mobile core). The configuration of these subscriptions is out of the scope of this scenario.

4. Now, when the Application receives a notification about the event **Account/Service/QuotaExceeded**, it connects to the ESPF via the API and sends a request to add the event **Custom/NextMonth** with the start time March 1, 2021, 00:00 and **i_account** attribute.

   NOTE: The start_time value displays time in the UTC time zone. When specifying the start_time value, convert the time of the customer's time zone to the UTC time.

   Here is an example of the HTTP POST request:

```
Authorization: Basic dXNlcm5hbWU6c2VjcmV0
Content-Type: application/json
POST /espf/v1/event/create HTTP/1.1

{
"event_type": " Custom/NextMonth ",
"variables": {
"i_account": 541
},
"start_time": "2021-03-01 00:00:00"
}
```

   Here is an example of the response:

```
{ "i_event": 623623 }
```

As a result, on March 1, the Application receives a request from ESPF with the event **Custom/NextMonth**. The Application retrieves the account details from PortaBilling® using the API method **Account/get_account_info**. If the application detects that the account has sufficient funds, it notifies the HSS to unblock this service for that specific account.

Now, the service is unblocked in the mobile core. When the account user attempts to access the service, e.g., on March 5, PortaBilling® receives the authorization request and allocates the quota for March. The Internet service in roaming is available for the user.

Note that there's a possible scenario when the account doesn't have sufficient funds at the beginning of the month, e.g., on March 1, but the user tops up their account later. In this case, additionally, configure the **Account/Service/QuotaAvailable** event type at step 3. After the top-up, the Application receives the notification about the **Account/Service/QuotaAvailable** event and notifies the HSS to unblock the service for this account.

Appendices

APPENDIX A. Authentication methods

PortaBilling® ESPF supports these HTTP authentication schemes:

Basic

The authentication is done 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

The authentication is done 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

The authentication is done authenticate by signature key and key ID.

Credentials string 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.

Bearer

Authentication is done using the JWT-encoded (JSON Web Token) access token.

The Authorization header is in the following format:

```
Authorization: Bearer <token>
```

An access token is an encrypted string, usually generated by the server in response to a login request and used in subsequent requests to protected resources (e.g. API). The "Bearer" scheme can be understood as "give access to the bearer of this token".

A JWT token consists of three base64-encoded parts separated by dots:
`<header>.<payload>.<signature>`

The **header** includes the token type (JWT) and the signing algorythm (e.g. HMACSHA256).

The **payload** contains the claims. This is the information about the user and additional data. In PortaBilling®, the claims are the `i_env` value and the token expiration time defined in the Unix Time format.

The **signature** part is used to verify that the message wasn't changed along the way and that the token sender is who they says they are. It includes the encoded header and payload, a signature key and is signed using the signing algorithm. A signature key is defined in the **BearerAuth.Key** option on the Configuration server.

To put it all together, let's say the signature key is *superkey*, the payload values are: *i_env=3* and the expiration time is *1560338716* (Wed Jun 12 14:25:16 2019 EEST). The signing algorithm is *HMACSHA256*. The authorization header then looks as follows:

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJpX2VudiI6MywiRXhwIjoxNTYwMzQyMzE2fQ.PONYERv3xYs5g--
XwbflOcaWddSuUNt7hAQqvoEcO4Q
```

The ESPF and the Application must know the signature key. When the Application receives the request, it:

- captures the value from the Authorization header;
- strips the "Bearer" prefix before the token;
- runs the same signature algorithm to receive the signature value;
- compares the obtained signature's own hashing operation with the signature on the token itself.

If the signatures match, authorization is successful.

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 Subcriber 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);
use Cache::LRU;

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 );

# track active requests to detect retries
my $active_req = Cache::LRU->new( size => 100 );
```

```
# 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;
} ## end sub api_login

# 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;
} ## end foreach my $sim ( @{$sim_list...})
} ## end else [ if ( scalar( @{$sim_list...}))]

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' )
);

# TEST: instert some random delay,
# to emulate request timeout on remote side
my $test_sleep = int( rand(10) );
log_debug( 'Emulate network delay, sleep ' . $test_sleep );
sleep($test_sleep);
```

```
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
log_error('Malformed JSON request');
return HTTP_BAD_REQUEST;
}

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

# detect retries for long-running requests
my $unique_id = $event->{variables}->{i_event};
if ( defined $unique_id
&& ( my $cached_result = $active_req->get($unique_id) ) ) {
log_debug(
sprintf 'Detected retry request #%d, result: %s',
$unique_id, $cached_result
);

if ( $cached_result ne '-' ) {
# remove stored result
# and return it without 'long' processing
$active_req->remove($unique_id);
return $cached_result;
}
else {
# request 'in-progress'. Depending on implementation
# it can wait for result or start new processing.
# For this example we restart processing
$active_req->remove($unique_id);
}
} ## end if ( defined $unique_id...)

# 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;
}

# store 'start' of processing
$active_req->set( $unique_id => '-' );

if (
!provision_external_system( {
action    => $action,
account   => $account,
```

```
sim_cards => $sim_card_list,
}
)
) {
# TODO add required error processing (alerts, retries, etc)
$active_req->remove($unique_id);
return HTTP_INTERNAL_SERVER_ERROR;
}

# store result
$active_req->set( $unique_id => HTTP_OK );

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;
```