AVEND Remote Vend API

v1.1.0 – AAEON Electronics

Revisions

Version	Date	Revision
1.0.0	2023-03-03	Original
1.0.1	2023-03-03	Enhanced formatting, added images
1.0.2	2023-03-03	Changed request method from GET to POST
1.0.3	2023-03-07	Added cover page; added ‘action’ parameter to API examples
1.0.4	2023-03-31	Enhanced ‘add’ method response
1.0.5	2023-09-11	Added optional timeout parameter
1.1.0	2023-09-12	Added revision table, timeout now uses seconds instead of minutes

Contents

Overview

AAEON provides the AVEND Remote Vend API (“API”) for comprehensive vending machine dispense operations. The API implements the HTTP REST model and uses a JSON format for requests and responses.

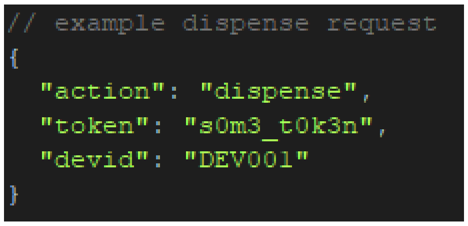
API URL

Endpoint

https://api.avend.us/remote_vend/

This is the endpoint that will be used to access the AVEND Remote Vend API. Each call to the API requires POST data sent in a JSON format, including an ‘action’ parameter that specifies which operation you would like to perform. Additional parameters required for each action are detailed in the following sections.

Example Request Body



```
// example dispense request
{
  "action": "dispense",
  "token": "s0m3_t0k3n",
  "devid": "DEV001"
}
```

API Methods

Auth

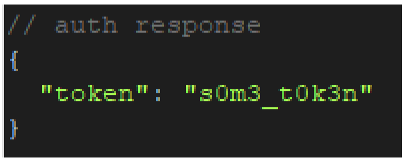
This method is used to obtain an authorization token for a new vending session.

Required Parameters

Key	Description
action	‘auth’
appkey	The application key provided by AAEON
devid	The device ID for the machine to be used for this session.
timeout	Optional parameter to specify token lifespan, in seconds. Allows values between 60-900 seconds.

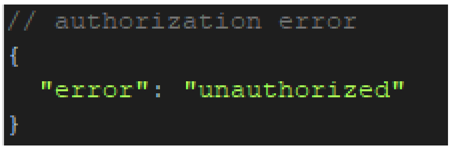
A successful authorization returns a token which can be used for subsequent API calls for the duration of the session (default is 300 seconds). A particular appkey / devid combination can only have one session at a time; subsequent calls to auth before a token has expired will return the currently active token.

Example Responses



```
// auth response
{
  "token": "s0m3_t0k3n"
}
```

Attempts to use the API without a token, or with an unauthorized token will result in an error:



```
// authorization error
{
  "error": "unauthorized"
}
```

Dispense

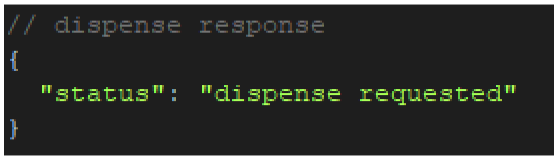
The dispense method has two different use cases.

In the first use case, it can be used to directly dispense a single item by including the desired column code in the request:

Required Parameters

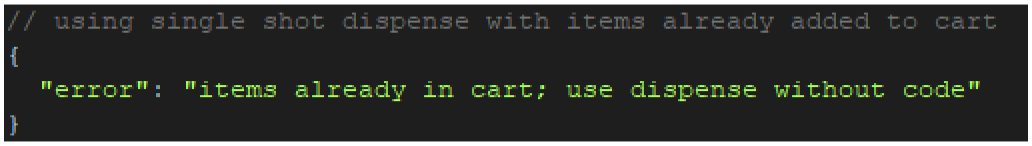
Key	Description
action	‘dispense’
token	The token provided by the ‘auth’ action
devid	The device ID for the machine to be used for this session.
code	This is the visible slot identifier and can be alphanumeric.

Example Responses



```
// dispense response
{
  "status": "dispense requested"
}
```

Attempts to use this single shot dispense while there are already items added to the cart will result in an error:



```
// using single shot dispense with items already added to cart
{
  "error": "items already in cart; use dispense without code"
}
```

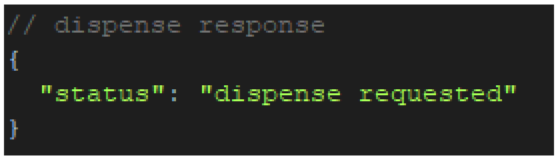
v1.1.0

In the second use case, you can dispense all items currently in the cart. This use case requires items to be added to the cart using the ‘add’ action.

Required Parameters

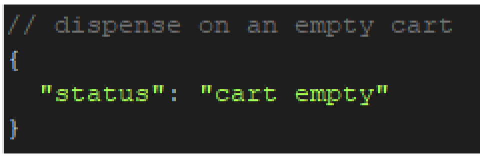
Key	Description
action	‘dispense’
token	The token provided by the ‘auth’ action
devid	The device ID for the machine to be used for this session.

Example Responses



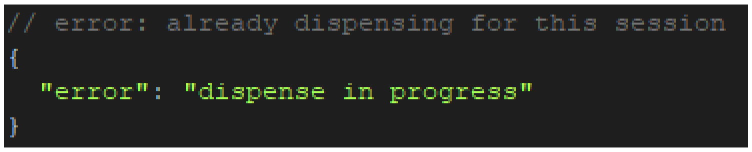
```
// dispense response
{
  "status": "dispense requested"
}
```

Attempting to call dispense when the cart is empty will return an empty cart status:



```
// dispense on an empty cart
{
  "status": "cart empty"
}
```

In both use cases, once dispense has been called for a particular session, further cart manipulation using add, remove, clear, or dispense will not be allowed:



```
// error: already dispensing for this session
{
  "error": "dispense in progress"
}
```

v1.1.0

Add

This method is used to add an item to the cart. The cart has a 5 item limit.

Required Parameters

Key	Description
action	‘add’
token	The token provided by the ‘auth’ action
devid	The device ID for the machine to be used for this session.
code	This is the visible slot identifier and can be alphanumeric.

Example Responses

```
// add response
{
  "added_item": "Some Product Name"
}
```

If there is no item data on the server, added_item will return a generic “Item #n”, where *n* is the slot position for the added item:

```
// add repsonse for item with no data on server
{
  "added_item": "Item #4"
}
```

Attempting to add another item once 5 items are already in the cart will result in an error:

```
// cart size limit reached
{
  "error": "cart size limit reached"
}
```

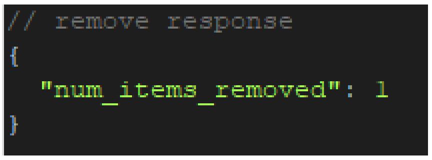
Remove

This method is used to remove an item from the cart.

Required Parameters

Key	Description
action	‘remove’
token	The token provided by the ‘auth’ action
devid	The device ID for the machine to be used for this session.
code	This is the visible slot identifier and can be alphanumeric.

Example Response



```
// remove response
{
  "num_items_removed": 1
}
```

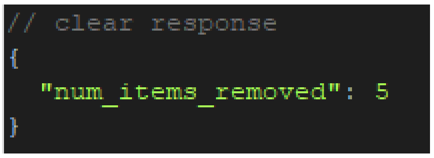
Clear

This method is used to clear all existing items from the cart.

Required Parameters

Key	Description
action	‘clear’
token	The token provided by the ‘auth’ action
devid	The device ID for the machine to be used for this session.

Example Response



```
// clear response
{
  "num_items_removed": 5
}
```

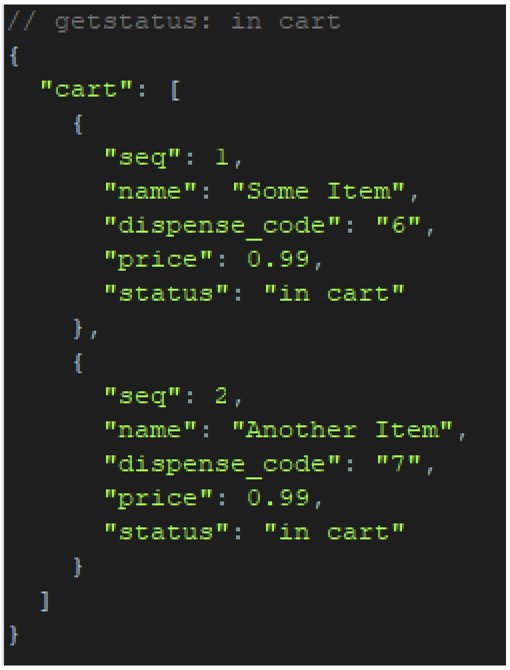
Get Status

This method is used to get the status for all items in the cart. It can be used at any point during the session.

Required Parameters

Key	Description
action	‘getstatus’
token	The token provided by the ‘auth’ action

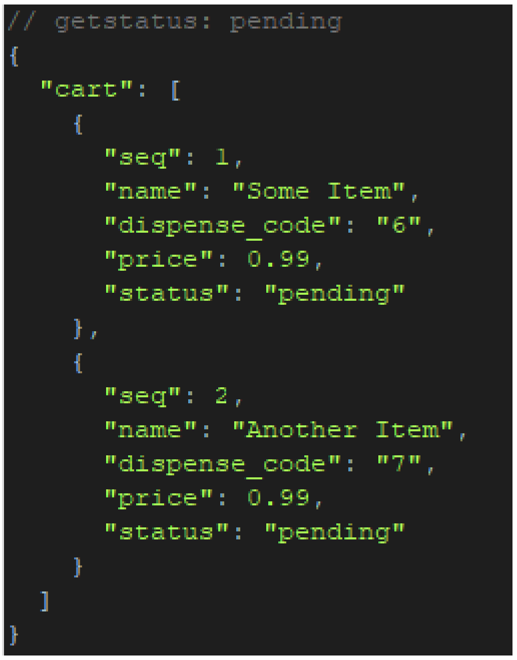
Example Responses



```
// getstatus: in cart
{
  "cart": [
    {
      "seq": 1,
      "name": "Some Item",
      "dispense_code": "6",
      "price": 0.99,
      "status": "in cart"
    },
    {
      "seq": 2,
      "name": "Another Item",
      "dispense_code": "7",
      "price": 0.99,
      "status": "in cart"
    }
  ]
}
```

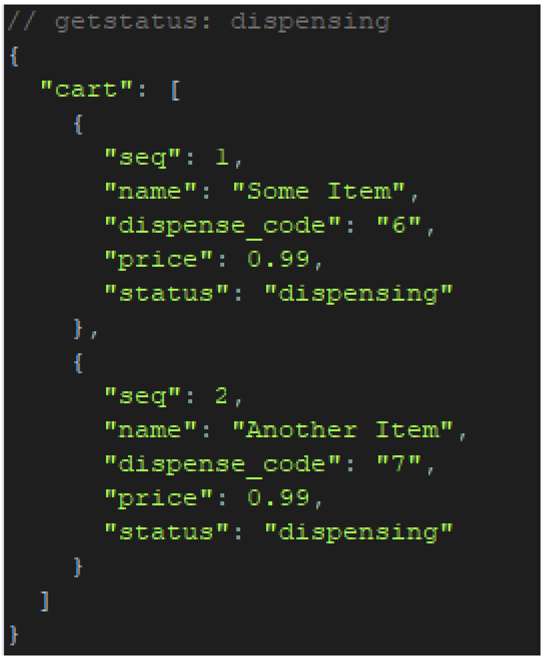
v1.1.0

Once a dispense is initiated, the status for each item will change to ‘pending’:



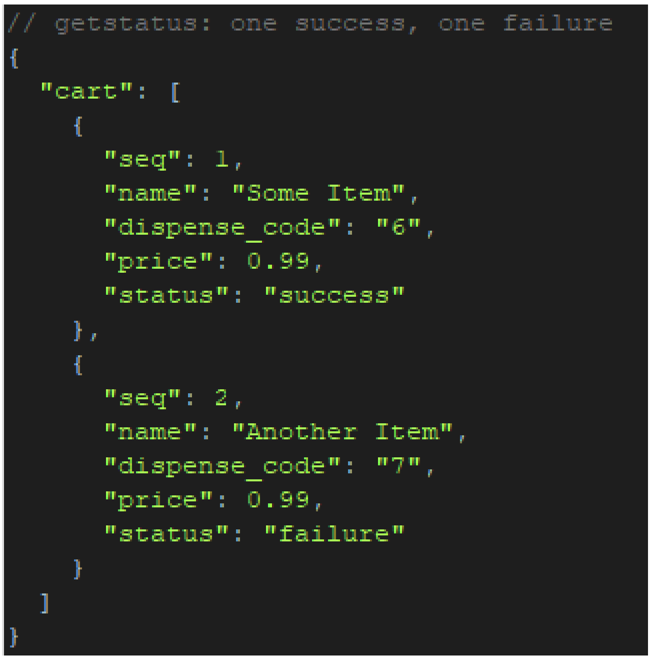
```
// getstatus: pending
{
  "cart": [
    {
      "seq": 1,
      "name": "Some Item",
      "dispense_code": "6",
      "price": 0.99,
      "status": "pending"
    },
    {
      "seq": 2,
      "name": "Another Item",
      "dispense_code": "7",
      "price": 0.99,
      "status": "pending"
    }
  ]
}
```

When the kiosk captures the dispense request, the status for each item will change to ‘dispensing’:



```
// getstatus: dispensing
{
  "cart": [
    {
      "seq": 1,
      "name": "Some Item",
      "dispense_code": "6",
      "price": 0.99,
      "status": "dispensing"
    },
    {
      "seq": 2,
      "name": "Another Item",
      "dispense_code": "7",
      "price": 0.99,
      "status": "dispensing"
    }
  ]
}
```

Once getstatus returns a cart for which each item’s status is “success” or “failure”, the cart is considered resolved:



```
// getstatus: one success, one failure
{
  "cart": [
    {
      "seq": 1,
      "name": "Some Item",
      "dispense_code": "6",
      "price": 0.99,
      "status": "success"
    },
    {
      "seq": 2,
      "name": "Another Item",
      "dispense_code": "7",
      "price": 0.99,
      "status": "failure"
    }
  ]
}
```

At this point, the session will be terminated, and the token will be invalidated. Subsequent calls to the Remote Vend API using the token will return as unauthorized.