# hiku External API Documentation v1.5.6

Nils Gura, Rajan Bala
hiku labs, inc

Change log

| Version | Changes |
|---|---|
| 1.5 | Introduced event API including PRODUCT_ENTRY, EAN_NOT_FOUND, and DEVICE_BATTERY_LEVEL events |
| 1.5.1 | Made fields “ean” and “audioPath” required fields for PRODUCT_ENTRY event to provide backward compatibility.<br>Updated data type for “batteryLevel” in DEVICE_BATTERY_LEVEL to be a string that represents an integer from 0-100. |
| 1.5.2 | Added Device registration event and user query api to return the email address of the user to third parties that are enabled for it |
| 1.5.3 | Added the api description to add items to a list on hiku cloud |
| 1.5.4 | Added api description to fetch and manipulate shopping lists |
| 1.5.5 | Added the aisle API description |
| 1.5.6 | Update the documentation to outline the list/PUT |

# API Basics

All requests go to baseUrl/api/apiVersion
base url = https://hiku.herokuapp.com/
API Version = 1
>> https://hiku.herokuapp.com/api/v1/

All requests must contain the following parameters
app_id = yourAppId
sig = a sha256 hash of (app_id+secret+time).
time = String version of the current utc time. The time field should follow ISO-8601 standard with the following format: “YYYY-MM-DD hh:mm:ss.SSSSSS”. If your programming language of choice does not support seconds to 6 decimal places, then set any trailing digits to zeros. Of course the time value should be the same for the time parameter and the time used in the signature. Note: while all parameters should be URL encoded in https requests, make sure you do *NOT* use the URL encoded time when generating the signature.

Email devrel@hiku.us to register for an app_id and secret.

All successful responses follow the format of

```
{
"response": {
        "status": "ok",
        "data": {
                "somedata": []
                }
        }
}
```

data is optional

Any HTTP status code may be returned based on service reachability, however all valid hiku service errors return HTTP 200 status with a payload as follows describing the error:

```
{
"response": {
        "status": "error",
        "errMsg": "some reason for the error"
        }
}
```

# Event API
# Receiving device data (aka beeps) and events

Any time a user scans or speaks an item with hiku, it is called a beep. These beeps trigger a corresponding event that can be forwarded to third parties using a URL webhook. In addition to the beep events, other events can selectively be forwarded such as device battery levels or notifications of unrecognized barcodes. To register your third party application webhook, send the URL and your app_id to devrel@hiku.us.

Once a user is registered for a third party app, all events the app is configured to receive trigger a webhook POST, i.e. the event data is sent to a URL specific to the app. The body of the POST request contains a data object in JSON format, where the data encoding can be configured per app to be either 'application/json' or 'application/x-www-form-urlencoded'. Event data objects have the following structure:

{"data" : {"eventId": <eventId>, "eventDesc": <eventDesc>, <required parameters>, <optional parameters>}}

Parameters eventId and eventDesc uniquely identify an event, where eventId is a positive integer specifying the event type and eventDesc is a string providing a human-readable description of the event. Each event type has required parameters, which are always present, and optional parameters, which may be present only in some event instances. Parameters may appear in the JSON object in any order. To allow for future extensions of the event API, implementations handling events should ignore additional, unrecognized parameters.

The table below lists the data types used by the event API, which are a subset of the JSON data types.

| **Type** | **JSON Type** | **Description** |
|---|---|---|
| Int | Number | 32-bit signed integer |
| String | String | ASCII string |
| UString | String | Unicode string |
| HString | String | String containing a hexadecimal number |

## Beeps/product entry event

Beeps/product entry events are generated whenever a user within the hiku system adds a product to a shopping list. Products can be entered with hiku through a barcode scan or voice entry, or within the hiku or third-party apps through textual entry, selection from favorites, barcode scan, or voice entry.

| Parameter | Req/Opt | Type | Description/Value |
|---|---|---|---|
| eventId | Req | Int | 1 |
| eventDesc | Req | String | PRODUCT_ENTRY |
| token | Req | HString | Unique token identifying the user in the third party system |
| id | Req | Int | Shoplist entry id of the product |
| name | Req | UString | Product name/description |
| ean | Req | String | Barcode/EAN, variable length. Set to the EAN when the item was scanned, set to NULL otherwise. |
| audioPath | Req | String | URL to a WAV file containing the voice recording for playback. Set when voice entry was used, set to the empty string otherwise. |
| serialNumber | Opt | HString | hiku serial number, only present if a hiku device was used for product entry |

EXAMPLES:

Product entry with barcode scan, format application/json
{"data": {"token": "f4fef8ad14f46595be4c3f62420b08294f68de320b5a2cbab367589a7022e549", "eventDesc": "PRODUCT_ENTRY", "name": "Caress Velvet Bliss Ultra Silkening Beauty Bar - 6 CT", "ean": "11111065925", “audioPath”: “”, "serialNumber": "46229f72bce7b25ee82aafb85bc65ede", "eventId": 1, "id": 8423}}
Product entry with voice, format application/x-www-form-urlencoded
data=%7B%22token%22%3A+%220c4ea3833226aee570bc35397cde804bdbcdbffeb52bcb682fc6ff9e2679df9f%22%2C+%22audioPath%22%3A+%22http%3A%2F%2Ftests3.hiku.us.s3.amazonaws.com%2Ftmp%2Fdb2b75f61cfd0fa7.wav%22%2C+%22eventDesc%22%3A+%22PRODUCT_ENTRY%22%2C+%22name%22%3A+%22Paper+towels%22%2C+%22serialNumber%2

2%3A+%2246229f72bce7b25ee82aafb85bc65ede%22%2C+%22eventId%22%3A+1%2C+%22ean%22%3A+NULL%2C+%22id%22%3A+8417%7D

## Barcode-not-found event

In some cases, a scanned barcode may not be found in the databases hiku sources from. This may be the case for private-label products or products otherwise not registered in publicly accessible databases. Apps can subscribe to the EAN_NOT_FOUND event to be notified of unrecognized barcodes.

| Parameter | Req/Opt | Type | Description/Value |
|---|---|---|---|
| eventId | Req | Int | 2 |
| eventDesc | Req | String | EAN_NOT_FOUND |
| token | Req | HString | Unique token identifying the user in the third party system |
| eanNotFound | Req | String | Unrecognized barcode, variable length |

EXAMPLES:

Barcode not found, format application/json

{"data": {"eventDesc": "EAN_NOT_FOUND", "eventId": 2, "token": "f4fef8ad14f46595be4c3f62420b08294f68de320b5a2cbab367589a7022e549", "eanNotFound": "111111111111"}}
Barcode not found, format application/x-www-form-urlencoded
data=%7B%22eventDesc%22%3A+%22EAN_NOT_FOUND%22%2C+%22eventId%22%3A+2%2C+%22token%22%3A+%220c4ea3833226aee570bc35397cde804bdbcdbffeb52bcb682fc6ff9e2679df9f%22%2C+%22eanNotFound%22%3A+%221111111111111%22%7D

## Battery level event

hiku devices periodically report their battery levels, which can be displayed in an app to remind users to charge their devices. The battery level is provided as a percentage, where 100% indicates a full charge and 0% indicates an empty battery. It is recommended that users be reminded to charge their devices once the battery level drops below 20%.
A battery level event is generated whenever a device reports its battery level and the value differs from the previously reported value.

| Parameter | Req/Opt | Type | Description/Value |
|---|---|---|---|
| eventId | Req | Int | 3 |
| eventDesc | Req | String | DEVICE_BATTERY_LEVEL |
| token | Req | HString | Unique token identifying the user/device owner in the third party system |
| batteryLevel | Req | String | Battery level, percentage from 0-100 as a string; can be converted to an integer value |
| serialNumber | Req | HString | hiku serial number |

EXAMPLES:
Battery level, format application/json
{"data": {"token": "f4fef8ad14f46595be4c3f62420b08294f68de320b5a2cbab367589a7022e549", "eventId": 3, "serialNumber": "62dfb41be54951c1", "eventDesc": "DEVICE_BATTERY_LEVEL", "batteryLevel": "1"}}
Battery level, format application/x-www-form-urlencoded
data=%7B%22token%22%3A+%220c4ea3833226aee570bc35397cde804bdbcdbffeb52bcb682fc6ff9e2679df9f%22%2C+%22eventId%22%3A+3%2C+%22serialNumber%22%3A+%2262dfb41be54951c1%22%2C+%22eventDesc%22%3A+%22DEVICE_BATTERY_LEVEL%22%2C+%22batteryLevel%22%3A+%221%22%7D

## Device registration level event

Some third party requires to have an event sent upon a successful hiku device registration. The following event would notify the webhook as soon as a device is successfully added to a user's account.

| Parameter | Req/Opt | Type | Description/Value |
|---|---|---|---|
| eventId | Req | Int | 4 |
| eventDesc | Req | String | DEVICE_REGISTERED |
| token | Req | HString | Unique token identifying the user/device owner in the third party system |
| serialNumber | Req | HString | hiku serial number |

EXAMPLES:
Device registration, format application/json
{"data": {"token": "f4fef8ad14f46595be4c3f62420b08294f68de320b5a2cbab367589a7022e549", "eventId": 4, "serialNumber": "62dfb41be54951c1", "eventDesc": "DEVICE_REGISTERED"}}
Device Registration, format application/x-www-form-urlencoded
data=%7B%27eventId%27%3A+4%2C+%27token%27%3A+%27f4fef8ad14f46595be4c3f62420b08294f68de320b5a2cbab367589a7022e549%27%2C+%27serialNumber%27%3A+%2762dfb41be54951c1%27%2C+%27eventDesc%27%3A+%27DEVICE_REGISTERED%27%7D

# /apps

**Note: ** This api method of enabling third party is deprecated and third parties now should utilize hiku SDK for iOS and Android to pair an account and setup hiku devices. Documents are found at: https://github.com/hikuinc/hiku_shared/tree/master/SDK**

## POST /apps - Add a third party app for a user

This is how a third party app initiates the process for adding their app to a user's hiku account. /apps POST returns a frob and barcode image. Present the barcode image to the user to scan with their hiku device. After scanning the barcode, the device is authenticated to the third party app and the device will emit the 'happy up-beep'.

The third party app then calls /apps PUT with the frob to receive a unique user token which will be sent with subsequent webhook POST's for that user.

Form-encoded POST parameters:
None

Sample response:
{
"response": {
"data": {
"frob": "aUniqueFrob",
"barcode": "svgBarcodedImageOfTheFrob"
},

```
        "status": "ok"
        }
}
```


### PUT /apps/<frob> - Update a third party app to authenticate it

This returns a token that uniquely identifies a user. This token should be stored by the third party application and mapped to their corresponding user, since it will be sent with beep data in the webhook.

For the frob, use the one returned from /apps POST.

Sample response:


```
{
"response": {
        "data": {
                "status": "Authorized"
                "token": "aUniqueUserToken"
                },
        "status": "ok"
        }
}
```


token is only present when ‘status’: ‘Authorized’. If the frob is still awaiting verification, it will return ‘status’: ’New’ and the token will not be present.


```
{
"response": {
        "data": {
                "status": "New"
                },
        "status": "ok"
        }
}
```


## hiku user control of Apps

Once a third party adds an app for a user, it will appear in the hiku mobile app under Settings / Apps. To stop data from flowing to the third party, the user simple unchecks the checkbox next

to the third party app. Checking the checkbox restarts data flow to the third party.

# /list

## GET /list - Retrieve all list items for a user.

By default returns items with both active and crossed off status.

GET parameters:
token - the session token

Sample response:


```
{
"response": {
        "data": {
                "list": [
                        {
                        "aisle_id": 70,
                        "audioPath": null,
                        "flFavorite": false,
                        "id": 35,
                        "listTags": [
                                9
                                ],
                        "name": "pumpkin seeds",
                        "product_id": 36,
                        "quantity": 2,
                        "status": 1
                        }
                ]
                },
        "productOrder": [
              {
                "aisle_id": 13519,
                "name": "No Aisle",
                "productOrder": [
                  14073,
                  14072,
                  14071
                ]
              },
```

```
                {
                  "aisle_id": 13520,
                  "name": "Baby",
                  "productOrder": [
                    14078,
                    14077
                  ]
                },
          "regulars": [
                {
                  "aisle_id": 1992,
                  "name": "Napkins",
                  "product_id": 975
                },
                {
                  "aisle_id": 1989,
                  "name": "Club soda",
                  "product_id": 978
                }
          ],
          "status": "ok"
          }
}
```


/list GET also includes a comprehensive list of all regulars, as well as the sort order of the current list by aisle.

NOTE: product_id uniquely identifies a product for that user, while id identifies the incarnation of that product in the users shoplist.

So when you want to update the status of the shoplist item, use the 'DELETE' method with id's.

## DELETE /list - Cross off, clear, or reactive list items

GET parameters:
token - the session token
id - comma separated list of id’s
action - accepts ‘clear’, ‘crossOff’, ‘reActivate’ (aka un-delete)

Sample response,  note no data payload is returned with /list DELETE:


```
{
"response": {
          "status": "ok"
```

```
        }
}
```

## POST /list - add items to the list

Although most integrators of our API have their own list management, there are some partners interested in leveraging our cloud to store the list.  Adding an item to a list is fairly straight-forward.

| Parameter | Req/Opt | Type | Description |
|---|---|---|---|
| **ean** | Optional if name is provided | String | Barcode of the product being added |
| **name** | Optional if ean is provided | String | Name of the product being added |
| **token** | Req | HString | Unique user token |
| **id** | Opt | Int | Shoplist ID if you already have a product on hiku shoplist |
| **aisle_id** | Opt | Int | hiku Aisle ID if you already know the aisle_id it belongs |
| **listTags** | Opt | Array of Ints | Shoplist IDs where that product should belong |
| **quantity** | Opt | Int | Quantity of the product you would like, default is 1 |
| **flFavorite** | Opt | True/False | If this item is one of your favorites |

A POST with the url encoded parameters as listed in the table above, the hiku cloud would yield a json response of the added item as follows should the add operation went successful:

```
{
  "response": {
    "data": {
      "aisleName": "Canned Foods/Soups/Mixes",
      "aisle_id": 51335,
      "audioPath": "",
      "ean": "0070492145008",
      "flFavorite": true,
      "id": 20323,
      "listTags": [
        3511
```

```
      ],
      "name": "Fortune Stir Fry Noodles Japanese Style, With Seasoning Sauce Base",
      "product_id": 19139,
      "quantity": 6,
      "status": 1
    },
    "status": "ok"
  }
}
```


## PUT /list/<product_id> - add items to the list

Although most integrators of our API have their own list management, there are some partners interested in leveraging our cloud to store the list.  Adding an item to a list is fairly straight-forward.

| Parameter | Req/Opt | Type | Description |
|---|---|---|---|
| **name** | Opt | String | new name for the product being updated |
| **token** | Req | HString | Unique user token |
| **aisle_id** | Opt | Int | hiku Aisle ID if you already know the aisle_id it belongs |
| **listTags** | Opt | Array of Ints | Shoplist IDs where that product should belong |
| **quantity** | Opt | Int | Quantity of the product you would like, default is 1 |
| **flFavorite** | Opt | True/False | Toggle on and off the favorite flag |
| **searchMethod** | Opt | String | By default the product is searched based on product ID provided in the put path.  We would also allow put by shoplist ID if search method is set as ‘byShoplistId” |

Upon successful put, you would receive the updated product detail as a JSON response.

# /listTags

## GET /listTags - Retrieves list of shopping lists

GET parameters:

token - the session token

Sample response:


```
{
  "response": {
    "data": {
      "forceDefault": false,
      "listTags": [
        {
          "aisleOrder": [
            52621,
            52622,
            52628,
            52624,
            52623,
            52625,
            52627,
            52626,
            52631,
            52634,
            52629,
            52630,
            52632,
            52633,
            52635,
            52636,
            52637,
            52638,
            52639,
            52640,
            52641,
            52642
          ],
          "defaultListTags": true,
          "id": 3653,
          "name": "Shopping list"
        }
      ]
    },
    "status": "ok"
  }
}
```

where the fields are as follows:
**id**: identifier of the shopping list
**defaultListTags**: indicates whether this shopping list is the default list to add
**name**: name of the shopping list
**aisleOrder**: the display order of the aisles within the shopping list
**forceDefault**: this switch is optional, if set to true, then adding an existing item after changing the default list would ensure that it adds it into the old list and the new list

### POST /listTags - Adds a new shopping list to the cloud

Most of the 3rd party would already have a shopping list by default, however should you decide to add a custom shopping list it can be added by doing a POST with url encoded parameters as follows:

| Parameter | Req/Opt | Type | Description |
|---|---|---|---|
| **name** | Req | String | Name of the new shopping list |
| **token** | Req | HString | Unique user token |

A post to the /listTags endpoint with the above mandatory fields as a url-encoded parameter, a new list would be added should there be no errors and would yield a json response such as the one below:


```
{
  "response": {
    "data": {
      "id": 3888,
      "name": "My New Shopping List"
    },
    "status": "ok"
  }
}
```


## /aisle

### GET /aisle - Retrieves the user's aisles

GET parameters:
token - the session token

Sample response:


```
{
  "response": {
    "data": {
      "aisle": [
        {
          "defaultAisle": false,
          "id": 54460,
          "name": "Baby"
        },
        {
          "defaultAisle": false,
          "id": 54461,
          "name": "Baking/Spices"
        },
        {
          "defaultAisle": false,
          "id": 54462,
          "name": "Beer/Wine/Liquor"
        },
        {
          "defaultAisle": false,
          "id": 54463,
          "name": "Bread/Baked Goods"
        },
        {
          "defaultAisle": false,
          "id": 54464,
          "name": "Breakfast/Cereal"
        },
        {
          "defaultAisle": false,
          "id": 54465,
          "name": "Canned Foods/Soups/Mixes"
        },
        {
          "defaultAisle": false,
          "id": 54466,
          "name": "Condiments/Dressings"
```

```
},
{
  "defaultAisle": false,
  "id": 54467,
  "name": "Dairy"
},
{
  "defaultAisle": false,
  "id": 54468,
  "name": "Deli"
},
{
  "defaultAisle": false,
  "id": 54469,
  "name": "Drinks"
},
{
  "defaultAisle": false,
  "id": 54470,
  "name": "Frozen"
},
{
  "defaultAisle": false,
  "id": 54471,
  "name": "Grains/Pasta"
},
{
  "defaultAisle": false,
  "id": 54472,
  "name": "Household/Cleaning"
},
{
  "defaultAisle": false,
  "id": 54473,
  "name": "Lawn/Garden"
},
{
  "defaultAisle": false,
  "id": 54474,
  "name": "Meats"
},
{
  "defaultAisle": false,
```

```
        "id": 54475,
        "name": "Paper/Plastic"
      },
      {
        "defaultAisle": false,
        "id": 54476,
        "name": "Pet Food/Supplies"
      },
      {
        "defaultAisle": false,
        "id": 54477,
        "name": "Pharmacy/Medicine"
      },
      {
        "defaultAisle": false,
        "id": 54478,
        "name": "Produce"
      },
      {
        "defaultAisle": false,
        "id": 54479,
        "name": "School/Office Supplies"
      },
      {
        "defaultAisle": false,
        "id": 54480,
        "name": "Snacks"
      },
      {
        "defaultAisle": true,
        "id": 54459,
        "name": "No Aisle"
      }
    ]
  },
  "status": "ok"
 }
}
```


## POST /aisle - Adds a new aisle

Adding a new Aisle is a simple post request with the name of the aisle as outlined in the table

below.

| Parameter | Req/Opt | Type | Description |
|---|---|---|---|
| **name** | Req | String | Name of the new Aisle |
| **token** | Req | HString | Unique user token |

Sample response:

```
{
  "response": {
    "data": {
      "id": 54481,
      "name": "Bathroom Supplies"
    },
    "status": "ok"
  }
}
```

### PUT /aisle/<aisle_id> - update an existing aisle

Sending a PUT request to the aisle endpoint with suffixed aisle_id would update the particular aisle attributes:

| Parameter | Req/Opt | Type | Description |
|---|---|---|---|
| **name** | Req | String | Name of the new Aisle |
| **token** | Req | HString | Unique user token |

DELETE /aisle/<aisle_id> - delete an existing aisle

Sending a DELETE request to the aisle endpoint with suffixed aisle_id would delete the aisle and move any items in that aisle to the default aisle (‘No Aisle’) for that user.

Delete parameter:
token - the session token

## /user

### GET /user - Retrieves the user detail

GET parameters:
token - the session token

Sample response:

```
{
"response": {
        "data": {
                “language”:”en-GB”
                “email”:”user@domain.co.uk”
                },
        "status": "ok"
        }
}
```

By default a fetch is performed on /user endpoint the response would just be the language.  For selected third parties the response would also include the email address of the user.  Should you need the information enabled, please contact devrel@hiku.us