Getting started with Node [version 1.8.2]

Using Dependency injections

To simplify things you could use our dependency injection right in your project and always be up to date with the latest changes.

```
npm install reception-system-sdk@verion --registry
https://pedidosya.jfrog.io/artifactory/api/npm/partner-integrations-npm-prod-local/
```

To use

```
require('reception-system-sdk');
let Credentials = require ('reception-system-sdk/lib/http/Credentials');
let Environments = require ('reception-system-sdk/lib/http/Environments');
let ApiClient = require ('reception-system-sdk/lib/ApiClient');
let PaginationOptions = require ('reception-system-sdk/lib/utils/PaginationOptions');
let OrderStatus = require ('reception-system-sdk/lib/models/OrderState');
```

- - - - - - - - - - - - - - - -

First of all you should decide which kind of integration to use. You have two types of possible integrations:

- **Centralized server:** The restaurants interacts with an application server.
- **Point to point:** The restaurants interacts directly with PedidosYa API.

Note: When you are using point to point method then you need an extra pair of credentials. This extra pair of credentials correspond to the restaurant itself.

Authentication

For accessing the API you need to create a *Credentials* object.

Centralized credentials example

```
let credentials = new Credentials();
credentials.clientId = 'your_client_id';
credentials.clientSecret = 'your_client_secret';
credentials.environment = Environments.DEVELOPMENT;
```

This kind of credentials will work for all the restaurants handled by the third party application.

Point to point credentials example

```
let credentials = new Credentials();
credentials.clientId = 'your_client_id';
credentials.clientSecret = 'your_client_secret';
credentials.username = 'restaurant_username';
credentials.password = 'restaurant_password';
credentials.environment = Environments.DEVELOPMENT;
```

This credentials will only work for the specified restaurant.

Environments

This SDK supports multiple environments:

- **Development:** Default environment, used for development only
- **Staging:** Pre-production environment, used before every release for final testings
- **Production:** Production environment, used for real life

Note: Please be sure that you deploy the application with the right environment.

Basic usage

Once you have the *Credentials* object created you must create an *ApiClient* object using the previous credential object.

```
let credentials = new Credentials();
credentials.clientId = 'your_client_id';
credentials.clientSecret = 'your_client_secret';
credentials.environment = Environments.DEVELOPMENT;

// With Promise
let api = new ApiClient(credentials);
  api.order.deliveryTime
    .getAll()
    .then(function(deliveryTimes) {
      console.log(deliveryTimes);
    })
    .catch(function(error) {
      console.log(error);
    });

// With async/await
async function getDeliveryTimes(api) {
  try {
    let deliveryTimes = await api.order.deliveryTime.getAll();
    console.log(deliveryTimes);
  } catch (error) {
    console.error(error);
  }
}
let api = new ApiClient(credentials);
let promise = getDeliveryTimes(api);
```

Important: You must instantiate the *ApiClient* at the very beginning of your application and save that instance of *ApiClient* (for ex: in an static variable) and use that instance in the whole life cycle of your application. Please do not instantiate the *ApiClient* inside a for loop for example.

Pagination

Some operations needs a PaginationOptions parameter. This parameter it's to handle the pagination. You have the possibility to use the default configuration for pagination invoking the method create:

```
PaginationOptions.create()
```

This method by default sets a limit of fifteen results and a offset equal to zero. If you want to specify the pagination options you can use:

```
PaginationOptions.create().withOffset(10).withLimit(50);
```

This configuration sets a limit of fifty results and a offset equal to ten.

You can use the method next to iterate over the pages:

```
let options = PaginationOptions.create();
let restaurants = [];
let newRestaurants = await api.restaurant.getAll(options);
restaurants.push(newRestaurants);
while (newRestaurants.length !== 0) {
  options.next();
  newRestaurants = await api.restaurant.getAll(options);
  restaurants.push(newRestaurants);
}
console.log(restaurants);
```

NOTE: The maximum value for the limit it's one hundred.

Orders operations

First of all, you must import our delivery times and rejection messages, what is a delivery time? and a rejection message?.

- **Delivery time:** Promised interval in which the order should be delivered.
- **Reject message:** If the restaurant can't accept the order then it must be rejected with one of the predefined reasons.

Important: *DeliveryTimes* and *RejectMessages* are dynamic, as well as their attributes, so do not harcode them.
A good practice is to import them every day or on application start up, and save them somewhere in your application.

You can get all the delivery times using this operation

```
let deliveryTimes = await api.order.deliveryTime.getAll();
```

List of delivery times

Code	Description ES	Description PT
1	Entre 15' y 30'	Entre 15' e 30'
2	Entre 30' y 45'	Entre 30' e 45'
3	Entre 45' y 60'	Entre 45' e 60'
4	Entre 60' y 90'	Entre 60' e 90'
5	Entre 90' y 120'	Entre 90' e 120'

6	24 horas	24 horas
7	48 horas	48 horas
8	72 horas	72 horas
9	Entre 120' y 150'	Entre 120' e 150'
10	12 horas	12 horas
11	Entre 150' y 180'	Entre 150' e 180'

List of reject messages

```
let rejectMessages = await api.order.rejectMessage.getAll();
```

There are two important attributes in *RejectMessage* that you have to consider: ForLogistics, ForPickup.
When rejecting an order that has the flag Logistics in true, you must not use/show reject messages that have ForLogistics in false.
The same rule applies to orders with the flag PickUp in true. The field to show in your system is *DescriptionES*.

Example:

```
let mySavedRejectMessages; // previously imported reject messages
let rejectMessagesForThisOrder = mySavedRejectMessages;
if (order.logistics) {
  rejectMessagesForThisOrder = rejectMessagesForThisOrder.filter(function(rm) {
    return rm.forLogistics;
  })
}
if (order.pickup) {
  rejectMessagesForThisOrder = rejectMessagesForThisOrder.filter(function(rm) {
    return rm.forPickup;
  })
}
```

Getting new orders
It's very simple to obtain new orders, you just have to call one operation and will do the job for you. This operation receives two callbacks, one for success and the other when some error occurs.

```
await api.order.getAll(
      null,
      PaginationOptions.create(),
      function onSuccess(order) {
        doSomething(order); // Process the order
        return true; // If the order is processed, false otherwise
      },
      function onError(error) {
        // Something went wrong
        console.log(error);
      }
```

```
    );
);
```

This operation handle the unexpected errors for you so it won't stop polling for new orders. In case of error you will receive the order again for reprocessing. In case you are using centralized keys, you will receive all the states of the order, the possible states are those described in *OrderState* class.

Note: Keep in mind that you will receive one order per callback so be aware that your import mechanism should not take so long.

Note: If you are a pharmacy partner you could receive an attachment that will contain the url for the image of the prescription. This is available as a list within the order.

Note: You could receive a company that will contain its name and document. This is available as a field within the user in an order.

REGULAR ORDERS

The client order for food when the restaurant it's opened. These kind of orders are for immediately delivery and they have the properties *Pickup*, *PreOrder* and *Logistics* in false value. The restaurant should delivery the order approximately at the time specified in *DeliveryDate*

PICKUP ORDERS

Some orders are not for delivery, instead the client will pick up the order in the restaurant. All the pick up orders have the property *Pickup* in *true*, *address* in *null* and the estimated pickup time in *PickupDate*.

ANTICIPATED ORDERS

The client can order for food before the restaurant it's opened, if that's the case then the *PreOrder* property has the value *true*. The restaurant should delivery the order at the time specified in *DeliveryDate*.

LOGISTICS ORDERS

This only applies if the restaurant delivery it's handled by PedidosYa's fleet, otherwise you should ignore this. All orders with logistics are identified by the property *Logistics* in *true*. In this case you must pay attention to the *PickupDate* property, this field should be static as contains the time that one rider will pick up the order to be delivered and the ETA Rider, a dynamic property that shows rider's timing to get to the pickup point.

ONLINE PAID ORDERS

If the order was paid with an online payment method then the field *online* in the *payment* property of the order is in *true*.

Orders with discounts

The Orders class contains a list of Discounts. When more than one discount are combined, the order in which they are applied is defined by "priority" (lower number, has priority). Fields "value" and "type" define discount calculation: "value" specifies the amount, and "type" indicates if it is a percentual or a fixed discount. For example:

- Type: Percent & Value: 20 => A 20% of the order amount is going to be discounted.
- Type: Amount & Value: 80 => $80 of the order amount is going to be discounted.

The Orders class contains a list of Discounts, with the corresponding priority.

Discounts apply to both the order amount and the shipping amount, and for each of these both the original amount (without discounts applied) and the final amount are included in the JSON:

- amountNoDiscount
- amount
- shippingNoDiscount

- shipping

We handle different kinds of discounts:

- **PROMOTION:** It is deprecated for now.
- **WEEKLY:** Discount available in certain days of the week for the entire catalogue - with the exception of products marked as promotion -. It’s a percentual discount, defined and paid by the partner.
- **JOKER:** Discount that allows the user to place an order with discount for a few minutes: the higher the amount, the greater the discount. For the purposes of the partner it is similar to a voucher (fixed amount), but the partner assumes the cost instead of PedidosYa.
- **PLUS_SHIPPING_COST:** If the user is in the Order and Plus program, shipping (fixed amount) is discounted, assumed by PedidosYa.
- **STAMPS:** 9 + 1, the brand assumes the cost of the average (fixed amount) of the last 9 orders that the user made to that branch.
- **VOUCHER:** Fixed amount in favor of the user, assumed by PedidosYa.
- **BINS:** Discount applied by the platform when using a specific card. It is a percentage. In BINES the discount is assumed by PedidosYa and then returned to the brand. The discount is a 25% with a limitation of up to $ 150. It is a business rule that is in the terms and conditions, for which they should add you in the loop before starting a campaign of this type for us to test it in STAGING.
- **SUBSIDIZED:** A discount (fixed amount) that applies over a specific product subsidized by PedidosYa.

Confirm an order

Once you processed the order and it's ready to be cooked you must confirm it indicating the estimated delivery time.

Note: An order must be confirmed before three minutes to achieve a good User Experience. In case an order is ignored (has no confirmation or rejection) by an integration for 15minutes after generated, it will be rejected automatically from our platform.

Note: For *Regular orders* and *Pick up orders* the method for confirmation it's the same, it's mandatory to specify the delivery time when the order it's going to be confirmed, otherwise you must confirm the order without a delivery time, if you specify one you are going to receive an error.

```
try {
    let deliveryTimeId = restaurantSelectedTime();
    let order = orderToConfirm();

    let result;
    if (order.preOrder || order.logistics) {
        result = await api.order.confirm(order);
    } else {
        result = await api.order.confirm(order, deliveryTimeId);
    }

    if (result) {
        confirmOrderLocally(order);
    }
} catch (error) {
    console.log(error);
    // Wait 5 seconds and retry again
    // Don't retry more than three times
}
```

Note: You can also use order and delivery time ids directly, there is an overloaded method for that.

Note: An order must be confirmed before three minutes.

Reject an order

If you can't process the order or the order cannot be delivered for some reason you should reject it.

```
try {
    let orderId = orderToReject();
    let rejectMessage = restaurantRejectReason();
    let result = await api.order.reject(order, rejectMessage);

    if (result) {
        rejectOrderLocally(orderId)
    }
} catch (error) {
  console.log(error);
  // Wait 5 seconds and retry again
  // Don't retry more than three times
}
```

Note: You can also use order and reject message ids directly, there is an overloaded method for that.

Also you can reject an order with a note

```
try {
  let orderId = orderToReject();
  let rejectMessageId = restaurantRejectReason();
  let result = await api.order.reject(orderId, rejectMessageId, "Rejection note");

  if (result) {
    rejectOrderLocally(orderId)
  }
} catch (error) {
  console.log(error);
  // Wait 5 seconds and retry again
  // Don't retry more than three times
}
```

Note: You can also use order and reject message ids directly, there is an overloaded method for that.

Dispatch an order

Once the order is cooked and it is ready to deliver, you must call this method to indicate the order is on the way to the client address.

```
try {
  let order = orderToDispatch();
  let result = await api.order.dispatch(order);

  if (result) {
    dispatchOrderLocally(order);
  }
} catch (error) {
  console.log(error);
}
```

Note: You can also use order and reject message ids directly, there is an overloaded method for that.

Retrieve orders of the day

It's possible to retrieve the last confirmed or rejected orders of the day if your are using point to point credentials. If you are using centralized credentials, you will receive the order again after changing the state, you should save it for future reference.

For example, to retrieve the last confirmed orders you should do it this way:

```
try {
  let pagination = PaginationOptions.create().withOffset(0).withLimit(5);
  let orders = await api.order.getAll(OrderState.CONFIRMED, pagination);

  if (orders) {
    doSomething(orders);
  }
} catch (error) {
  console.error(error);
}
```

In case you want the rejected ones just change *OrderState.CONFIRMED* with *OrderState.REJECTED*

Note: This will not return **all** the orders, the response is limited to the last fifteen (this is explained in the Pagination section). A good practice it's to save the the order id in your system.

If you have the order id then you can ask for a particular order.

```
try {
  let orderId = OrderId();
  let order = await api.order.get(orderId);
  if (order) {
    doSomething(order);
  }
} catch (error) {
  console.error(error);
}
```

Retrieve order tracking info for Logistics orders

It's possible to retrieve the order tracking info for Logistics orders. This could be used to have a more accurate date for which the driver retrieves the order from the store. This is useful for updating the ETA Rider and order's deliveryDate fields. The available information is :

- The driver's coordinates, ETA rider and the driver's name.
- The pickup date: Date for when the driver picks up the order at the store. Updated until the state is PREPARING (included). With this data it's possible to update the order's pickupDate.
- The estimated delivery date: Estimated date for when the driver arrives at the user's destination and delivers the order. Updated until DELIVERED state. This field corresponds with the order's deliveryDate field and that's the one that should be updated. The first time the tracking info is retrieved, it's probable that the order's deliveryDate coincides with this field. With this data it's possible to update the order's deliveryDate.
- The state of the tracking: These are the possible states for the order tracking:

- FAILURE: State of the tracking when it failed. The rest of the fields will be null.
- REQUESTING_DRIVER: A driver is being requested to pickup and deliver the order. The pickupDate and the estimatedDeliveryDate will be equal to the order's pickupDate and deliveryDate fields respectively.
- TRANSMITTING: The order is being sent to your system. All the information is available.
- TRANSMITTED: The order was finally sent to your system. All the information is available.
- PREPARING: The order is being prepared. All the information is available.
- DELIVERING: The order is being delivered by a driver. Pickup date is null.
- DELIVERED: The order was finally delivered. All data is null.
- CLOSED: The order is closed and no live tracking happens. All data is null.

The pickup date is mandatory to show in an order, as is the time that the order needs to be ready for our rider. It’s accepted with order’s confirmation. The estimated delivery date is a necessary value to update in the order. The driver’s name, ETA Rider and coordinates are part of the additional information attached to the order tracking info.

To do that, you should do it in this way:

```
try {
  let orderId = OrderId();
  let orderTracking = await api.order.tracking(orderId);
  if (orderTracking) {
      trackingState = orderTracking.state
      // driver info
      driver = orderTracking.driver
      // location info, lat & lng attributes for the driver's coordinates
      location = driver.location
      if (trackingState === TrackingState.PREPARING) {
          updatePickupDate(orderTracking);
      }
  }
} catch (error) {
  console.error(error);
}
```

Restaurants operations

Through RestaurantClient class you have some operations to handle closures and openings. You can close a restaurants through his id and a specific range of dates (in your country time). And you can open a restaurant previously closed (**you can only open restaurant's closed by the operation close**) from a certain date

Getting the restaurants

```
let restaurants = await api.restaurant.getAll(PaginationOptions.create());
```

Note: with this operation you can figure out the necessary id to invoke the open and close methods. The Restaurant also have the integrationCode, you can match you integrationCode with the PedidosYa id for the restaurant.

Close a restaurant

```
try {
  let restaurantId = restaurantIdToClose();
  let from = moment(new Date()).format('YYYY-MM-DDTHH:mm:ss');
  let to = moment(new Date()).add(60, 'm').format('YYYY-MM-DDTHH:mm:ss');
  await api.restaurant.close(restaurantId, from, to, 'Restaurant closed by problems');
} catch (error) {
  console.log(error);
}
```

Note: the 'to' parameter needs to be after the 'from' in this previous example.

Open a restaurant

```
let restaurantId = restaurantIdToClose();
let from = moment(new Date()).format('YYYY-MM-DDTHH:mm:ss');
await api.restaurant.open(restaurantId, from);
```

Note: this method is going to open the restaurant starting now, so if you call the method close with a date in the future, this request for close is going to be deleted.

Menus operations

Sections

Create section

It's possible to create a new section group within your existing menu. You will need the following fields:

- name (Mandatory, cannot be empty or null)
- integrationCode (Mandatory, cannot be empty or null)
- enabled (true/false)
- integrationName
- index

```
function newSection() {
    return {
      integrationCode: generateIntegrationCode(),
      integrationName: generateIntegrationName(),
      index: 1,
      enabled: true,
      name: generateIntegrationName()
    };
  }
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```

   try {
     let api = new ApiClient(getPointToPointIntegrationCredentials());
     assert(await api.menu.section.create(newSection()));
   } catch (error) {
     assert.fail('Can not create section: ' + error.message);
   }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to create the section.

- With centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
    assert(await api.menu.section.create(newSection(), 17217));
  } catch (error) {
    assert.fail('Can not create section: ' + error.message);
  }
  ```

- With centralized credentials and restaurant code

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
    assert(await api.menu.section.create(newSection(), 'C121'));
  } catch (error) {
    assert.fail('Can not create section: ' + error.message);
  }
  ```

  Update Section

  You can update the following fields of a section:

- name (Mandatory, cannot be empty or null)
- integrationCode (Mandatory, cannot be empty or null)
- integrationName
- enabled
- index

notes: The Section must exist.

```
function modifedSection() {
   return {
     integrationCode: getIntegrationCode(),
     integrationName: generateIntegrationName(),
     index: 1,
     enabled: true,
     name: generateIntegrationName()
   };
 }
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let api = new ApiClient(getPointToPointCredentials());
      assert(await api.menu.section.modify(modifedSection()));
    } catch (error) {
      assert.fail('Can not modify section: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to create the section.

- With centralized credentials and restaurant Id

```
    try {
      let api = new ApiClient(
        getIntegrationCentralizedCredentials()
      );
      let section = modifedSection();
      section.integrationName = 'NEW_INTEGRATION_NAME_' + getRandomInt(10000000);
      assert(await api.menu.section.modify(section, 17217));
    } catch (error) {
      assert.fail('Can not modify section: ' + error.message);
    }
```

- With centralized credentials and restaurant code

-
  ```
      try {
        let api = new ApiClient(
          getIntegrationCentralizedCredentials()
        );
        let section = modifedSection();
        section.integrationName = 'NEW_INTEGRATION_NAME_' + getRandomInt(10000000);
        assert(await api.menu.section.modify(section, 'C121'));
      } catch (error) {
        assert.fail('Can not modify section: ' + error.message);
      }
  ```

Delete section

You will need the following fields to delete a Section:

- name (Mandatory, cannot be empty or null)
- integrationCode (Mandatory, cannot be empty or null)
- integrationName
- index
- enabled

notes: The section must exist.

```
function newSection() {
    return {
      integrationCode: generateIntegrationCode(),
      integrationName: generateIntegrationName(),
      index: 1,
```

```
      enabled: false,
      name: generateIntegrationName()
    };
  }
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let api = new ApiClient(getPointToPointCredentials());
      let section = newSection();
      await api.menu.section.create(section);
      assert((await api.menu.section.delete(section)) === true);
    } catch (error) {
      assert.fail('Can not delete section: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to delete the section.

- With centralized credentials and restaurant Id

```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
    let section = newSection();
    await api.menu.section.create(section, 17217);
    assert((await api.menu.section.delete(section, 17217)) === true);
  } catch (error) {
    assert.fail('Can not delete section: ' + error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
    let section = newSection();
    await api.menu.section.create(section, 17217);
    assert((await api.menu.section.delete(section, 'C121')) === true);
  } catch (error) {
    assert.fail('Can not delete section: ' + error.message);
  }
```

Get all sections

You won't need any specific field to get all the menus sections.

Note: As response, you will receive a list of Sections, each of them with the following attributes:

- name

- integrationCode
- integrationName
- enabled
- index

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
  let sections = await api.menu.section.getAll();
  assert(sections.length > 0);
} catch (error) {
  assert.fail('Can not get all sections: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get all sections.

- With centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
     let sections = await api.menu.section.getAll(restraurantId);
     assert(sections.length > 0);
  } catch (error) {
    assert.fail('Can not get all sections: ' + error.message);
  }
  ```

- With centralized credentials and restaurant code

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
    let sections = await api.menu.section.getAll(restaurantCode);
    assert(sections.length > 0);
  } catch (error) {
    assert.fail('Can not get all sections: ' + error.message);
  }
  ```

Get a Section by name or integrationCode

If you want to get a Section by name, you will need the following field:

- name

If you want to get a Section by integrationCode, you will need the following field:

- integrationCode

Note: As response, you will receive a Section's fields (listed in getAll method), and also the following Products fields

- name,
- integrationCode,
- integrationName,
- index,
- enabled,
- price,
- description,
- image

```
function getSection() {
    return {
      name: 'NEW_SECTION',
      integrationCode: '787878'
    };
  }
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      await api.menu.section.getByName(getSection());
    } catch (error) {
      assert.fail('Can not get section: ' + error.message);
    }
```

```
    try {
      await api.menu.section.getByIntegrationCode(getSection());
    } catch (error) {
      assert.fail('Can not get section: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a section by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- Get by name or integration code with centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let section = getSection();
    await api.menu.section.getByName(section, restaurantId);
  } catch (error) {
    assert.fail('Can not get section: ' + error.message);
  }
  ```

  ```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
  ```

```
    );
    let section = getSection();
    await api.menu.section.getByIntegrationCode(section, restaurantId);
  } catch (error) {
    assert.fail('Can not get section: ' + error.message);
  }
```

- Get by name or integration code with centralized credentials and restaurant code

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let section = getSection();
    await api.menu.section.getByName(section, restaurantCode);
  } catch (error) {
    assert.fail('Can not get section: ' + error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let section = getSection();
    await api.menu.section.getByIntegrationCode(section, restaurantCode);
  } catch (error) {
    assert.fail('Can not get section: ' + error.message);
  }
```

Section schedules

Through the SectionClient class you have some operations to handle the configuration of its schedules. You can create, delete and get all the schedules of a particular section.

Create schedule

To create a schedule it is necessary to do it through the integration code or name of the section, and a specific range of time (in the time of your country) which is made up of the fields: to, from and day. For example:

```
function getSchedule() {
    return {
      to: '16:00',
      section: {
          name: 'Drinks'
      },
      from: '15:15',
      day: 4
    };
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
  await api.menu.section.createSchedule(getSchedule());
} catch (error) {
  assert.fail('Can not create section schedule: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a section by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- With centralized credentials and restaurant Id

```
try {
  let api = new ApiClient(
    SDKTestUtil.getIntegrationCentralizedCredentials()
  );
  let schedule = getSchedule();
  await api.menu.section.createSchedule(schedule, restaurantId);
} catch (error) {
  assert.fail('Can not create section schedule: ' + error.message);
}
```

- With centralized credentials and restaurant Code

```
try {
  let api = new ApiClient(
    SDKTestUtil.getIntegrationCentralizedCredentials()
  );
  let schedule = getSchedule();
  await api.menu.section.createSchedule(schedule, restaurantCode);
} catch (error) {
  assert.fail('Can not create section schedule: ' + error.message);
}
```

Note:

- For the field "to" and "from" must be Strings and follow the format: HH:mm in 24 hour time system.
- For the field "day" it must be an integer guided by the following equivalence:

Code | Name |:-- |:-------|
1 | Sunday 2 | Monday 3 | Tuesday 4 | Wednesday 5 | Thursday 6 | Friday 7 | Saturday

- For the schedules to be impacted, minutes must be multiples of 15. That is: 00, 15,30,45.
- Multiple schedules can be created for the same day, as long as they do not overlap each other. For example: if there is a schedule "from": "14:00" "to": "15:00", you can NOT create another like "from":"14:15", "to":"16:00".
- Also, it is necessary "from" < "to"

Delete schedule

You can delete a previously created schedule by sending the same information as when you create.

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
  await api.menu.section.deleteSchedule(getSchedule());
} catch (error) {
  assert.fail('Can not delete section schedule: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a section by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- With centralized credentials and restaurant Id

```
try {
  let api = new ApiClient(
    SDKTestUtil.getIntegrationCentralizedCredentials()
  );
  let schedule = getSchedule();
  await api.menu.section.deleteSchedule(schedule, restaurantId);
} catch (error) {
  assert.fail('Can not delete section schedule: ' + error.message);
}
```

- With centralized credentials and restaurant Code

```
try {
  let api = new ApiClient(
    SDKTestUtil.getIntegrationCentralizedCredentials()
  );
  let schedule = getSchedule();
  await api.menu.section.deleteSchedule(schedule, restaurantCode);
} catch (error) {
  assert.fail('Can not delete section schedule: ' + error.message);
}
```

Get all schedules

You can get all the section schedules sending only the name of it.

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
  await api.menu.section.getAllSchedules(getSchedule());
} catch (error) {
  assert.fail('Can not get all sections: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a section by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- With centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let schedule = getSchedule();
    await api.menu.section.getAllSchedules(schedule, restaurantId);
  } catch (error) {
    assert.fail('Can not get all sections: ' + error.message);
  }
  ```

- With centralized credentials and restaurant Code

  ```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let schedule = getSchedule();
    await api.menu.section.getAllSchedules(schedule, restaurantCode);
  } catch (error) {
    assert.fail('Can not get all sections: ' + error.message);
  }
  ```

Important: If you need to modify a schedule, you need to delete and recreate it.

###Product **Important:** If your product contains a barcode, set it in the gtin field. If your business is groceries, pharmacy, drinks or kiosk, the gtin field is mandatory. In the case that the product does not have a barcode, add the integrationCode in the gtin and integrationCode field.

If product mapping it's supported, then you should be able to handle stock and price changes. It's possible to disable, enable and change the price of products and optionals. The change will be immediately visible on our website and apps. If you want to disable a product that came mapped in the *Product* class through the *IntegrationCode* field then you need to use the *Menu.Product* operations, otherwise if it's an option also mapped through the *IntegrationCode* field then you have to use the *Menu.Option* operations. For example:

```
let order = someOrder();
let productCode = order.details[0].product.integrationCode;
// Important: restaurantId must be a number and restaurantCode a string
let restaurantId = order.restaurant.id;
let restaurantCode = order.restaurant.integrationCode;

await api.products.disable(productCode);
await api.products.disable(productCode, restaurantId); // Using restaurant id
await api.products.disable(productCode, restaurantCode); // Using restaurant code
```

Note: If a product is also mapped as an option then you should call the operations separately.

Create a product

It's possible to create a new product within your existing menu. The first thing you have to do is create a Section with the corresponding section name. Once you do that, you can create a new Product. The fields you could use are the following:

- name
- integrationCode (the code that will represent the product in your system)
- gtin (product barcode or the code that will represent the product in your system)
- integrationName (not mandatory)
- price (should be >0)
- image (url of the image of the product)
- description (not mandatory)
- enabled (true or false)
- index
- requiresAgeCheck (Indicates if this product requires age check. E.g.: cigarretes.)
- measurementUnit (Measurement unit of the Product.)
- contentQuantity (Content quantity of the Product expressed in its unit of measure. E.g. 1000 milliliters)
- prescriptionBehaviour (A prescription may be required, optional or not needed. E.g.: antibiotics)

```
 let section = {
    name: 'Test Products Section'
  };
  let product = {
    integrationCode: '992254',
    gtin: '7730400001729',
    name: 'Test Pizza',
    description: 'Tastiest test pizza',
    price: 980.99,
    image:

'https://cdn.apartmenttherapy.info/image/fetch/f_auto,q_auto:eco/https%3A%2F%2Fstorage.g
atmedia%2F3%2F2018%2F03%2F55cd28cae8ee78fe1e52ab1adc9eafff24c9af92.jpeg',
    section: section,
    enabled: true,
    index: 2
  };
```

notes: When creating the product the index will not be taken into account because it will be created in the last position.

Verify that the image url ends with a valid image type, for example jpg, png etc.

After creating the product you can add the previously created section as another field. The section is mandatory.

You have a few ways to create a product whether you have a centralized or point to point integration.

Suggestion: Check this guide [Partner Photography Guide](#) for PedidosYa best practices for images before uploading them.

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      assert(await apiClientToken.menu.products.create(product, null));
    } catch (error) {
      assert.fail('Can not create product with token: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to create the product.

- With centralized credentials and restaurant Id

  ```
    try {
        await apiClient.menu.products.create(product, restaurantId);
    } catch (error) {
      assert.fail('Can not create product with restaurant id: ' + error.message);
    }
  ```

- With centralized credentials and restaurant code

  ```
    try {
        await apiClient.menu.products.create(product, restaurantCode);
    } catch (error) {
      assert.fail('Can not create product with restaurant code: ' +
  error.message);
    }
  ```

  Another thing to take into consideration is that partners that belong to the new verticals such as groceries, pharmacy and drinks for the time being are not allowed to create new products.

Take into account, that the product you are creating is not going to map exactly to the product you receive in an order.

Modify a product

It’s also possible to edit an existing product. To be able to do this you need the products’ integration code. The possible fields that you can edit are:

- name
- integrationName (not mandatory)
- price (should be >0)
- image (url of the image of the product)
- description (not mandatory)
- enabled (true or false)
- index
- requiresAgeCheck (Indicates if this product requires age check. E.g.: cigarretes.)
- measurementUnit (Measurement unit of the Product.)
- contentQuantity (Content quantity of the Product expressed in its unit of measure. E.g. 1000 milliliters)
- prescriptionBehaviour (A prescription may be required, optional or not needed. E.g.: antibiotics)

Remember that the **price should be greater than 0** and the **image should end in a valid image type such as jpg**, png etc.

The index will be taken into account if the section the product is located does not have a subsided or outstanding product.

Another thing to take into consideration is that partners that belong to the new verticals such as groceries, pharmacy and drinks for the time being are not allowed to modify existing products.

Take into account, that the product you are modifying is not going to map exactly to the product you receive in an order.

```
 let section = {
    name: 'Test Products Section'
  };
  let modifiedProduct = {
    integrationCode: '992254',
    gtin: '7730400001729',
    name: 'Test Pizza Modified',
    description: 'Tastiest test pizza Modified',
    price: 1000,
    image:

'https://cdn.apartmenttherapy.info/image/fetch/f_auto,q_auto:eco/https%3A%2F%2Fstorage.g
atmedia%2F3%2F2018%2F03%2F55cd28cae8ee78fe1e52ab1adc9eafff24c9af92.jpeg',
    section: section,
    enabled: true,
    index: 2
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      await apiClientToken.menu.products.modify(modifiedProduct, null);
    } catch (error) {
      assert.fail('Can not modify product with token: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to modify the product.

- With centralized credentials and restaurant Id

```
  try {
        await apiClient.menu.products.modify(modifiedProduct, restaurantId);
  } catch (error) {
    assert.fail('Can not modify product with restaurant id: ' + error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
        await apiClient.menu.products.modify(modifiedProduct, restaurantCode);
  } catch (error) {
    assert.fail('Can not modify product with restaurant code: ' +
error.message);
  }
```

Delete a product

It’s possible to delete an existing product. To be able to do this you need the products’ integration code and section's integrationCode or name.

notes: You will not be able to delete a subsided or outstanding product.

```
    let section = {
    name: 'Test Products Section',
    integrationCode: 'ABC123'
    };
  let productToDelete = {
      section: section,
      integrationCode: '2001d3da-fe63-424e-bd2c-47b4ec490748',
      name: 'Test Pizza'
    };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      await apiClientToken.menu.products.delete(productToDelete, null);
    } catch (error) {
      assert.fail('Can not delete product with token: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to delete the product.

- With centralized credentials and restaurant Id

```
  try {
      await apiClient.menu.products.delete(productToDelete, restaurantId);
  } catch (error) {
    assert.fail('Can not delete product with restaurant id: ' + error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
      await apiClient.menu.products.delete(productToDelete, restaurantCode);
  } catch (error) {
    assert.fail('Can not delete product with restaurant code: ' +
error.message);
  }
```

Get all products

To get all products you will need the following field:

- section (you need it's integrationCode)

```
function getProducts() {
    let product = {
      section: { integrationCode: 'ABC' }
    };
    return product;
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let products = await api.menu.products.getAll(getProducts());
      assert(products.length > 0);
    } catch (error) {
      assert.fail('Can not get all products: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get all options group.

- With centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
     let products = await api.menu.products.getAll(getProduct(), restaurantId);
     assert(products.length > 0);
  } catch (error) {
    assert.fail('Can not get all products: ' + error.message);
  }
  ```

- With centralized credentials and restaurant code

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
     let products = await api.menu.products.getAll(getProduct(),
  restaurantCode);
     assert(products.length > 0);
  } catch (error) {
    assert.fail('Can not get all products: ' + error.message);
  }
  ```

Note As response, you will receive a list of Products

Get an Option Group by integrationCode

If you want to get the Product by integrationCode:

- integrationCode (the code that will represent the product in your system).
- section (you need at least it's integrationCode)

```
function getProduct() {
    let product = {
      integrationCode: 'ABC',
      name: 'Acompañamientos',
      section: { integrationCode: 'ABC' }
    };
    return product;
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let response = await api.menu.products.getByIntegrationCode(getProduct());
      assert(response);
    } catch (error) {
      assert.fail('Can not get product: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a option group by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- Get by integration code with centralized credentials and restaurant Id

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let response = await api.menu.products.getByIntegrationCode(getProduct(),
restaurantId);
    assert(response);
  } catch (error) {
    assert.fail('Can not get product: ' + error.message);
  }
```

- Get by integration code with centralized credentials and restaurant code

- With centralized credentials and restaurant code

```
 try {
   let api = new ApiClient(
     SDKTestUtil.getIntegrationCentralizedCredentials()
   );
   let response = await
api.menu.products.getByIntegrationCode(getProduct(),restaurantCode);
   assert(response);
 } catch (error) {
   assert.fail('Can not get product: ' + error.message);
 }
```

- *notes:** You are not able to retrieve a product by its' name.

###Optionals

Create optional group

It’s possible to create a new optional group within your existing product. The first thing you have to do is create a Product with the corresponding integrationCode and if you have the productId. Once you do that, you can create a new Optional Group. The fields you will need are the following:

- integrationCode (mandatory the code that will represent the optionGroup in your system).
- name (mandatory).
- integrationName
- product (The product which the optional group will belong to, integrationCode is mandatory)
- minimunQuantity
- maximunQuantity

notes: You have a few ways to operate with the values of minimumQuantity and maximumQuantity: for example, take into consideration the following: | Sent | Shown | |:------------- |:------------------| | Max 10, Min 2 | Max 10, Min 2 | | ONLY Max 10 | Fixed quantity 10 | | Max 10, Min 10| Fixed quantity 10 | | Max 0, Min 0 | Unlimited | | NO Max, NO Min| Unlimited |

notes: When maximunQuantity and minimunQuantity are the same, this is shown as fixed quantity on our site. maximunQuantity should be > than minimunQuantity

notes: The product integrationCode in your optional group is mandatory, you may add the productId if you have

notes: You may not have multiple optional groups with the same integrationCode or integrationName

notes: When creating the option Group the index will not be taken into account because it will be created in the last position.

```
let product = {
  integrationCode: 'INT987'
};

let optionGroup = {
  integrationCode: 'ABC123',
  name: 'Acompañamientos',
  product: product,
  maximumQuantity: 10,
  minimumQuantity: 2
};
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
    assert(await apiClientToken.menu.optionGroup.create(optionGroup));
} catch (error) {
    assert.fail('Can not create option group with token: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to create the optional group. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it's mandatory.

- With centralized credentials and restaurant Id

```
  try {
      assert(await apiClientToken.menu.optionGroup.create(optionGroup,
restaurantId));
  } catch (error) {
      assert.fail('Can not create optional group with restaurant id: ' +
error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
      assert(await apiClientToken.menu.optionGroup.create(optionGroup,
restaurantCode));
  } catch (error) {
      assert.fail('Can not create optional group with restaurant code: ' +
error.message);
  }
```

Update optional group

To update the optional group you need to follow the steps to create one, this means creating the product and following the restrictions mentioned before. The fields you will able to update are:

- integrationCode (the code that will represent the optionGroup in your system).
- name
- integrationName
- minimunQuantity
- maximunQuantity

notes: The product included in the optional group must exist. **notes:** The index will be taken into account if the section the product of the optional group is located does not have a subsided or outstanding product.

```
    let product = {
      integrationCode: 'INT987'
    };

    let optionGroup = {
      integrationCode: 'VDE323',
      name: 'Extras',
      product: product,
      maximumQuantity: 10,
      minimumQuantity: 5
    };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
        assert(await apiClientToken.menu.optionGroup.modify(optionGroup));
    } catch (error) {
        assert.fail('Can not modify option group with token: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to modify the optional group. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it's mandatory.

- With centralized credentials and restaurant Id

```
  try {
      assert(await apiClientToken.menu.optionGroup.modify(optionGroup,
restaurantId));
  } catch (error) {
      assert.fail('Can not modify optional group with restaurant id: ' +
error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
      assert(await apiClientToken.menu.optionGroup.modify(optionGroup,
restaurantCode));
  } catch (error) {
      assert.fail('Can not modify optional group with restaurant code: ' +
error.message);
  }
```

Delete optional group

To delete the optional group you need to follow the steps to create one, this means creating the product and following the restrictions mentioned before. The fields you will need are:

- integrationCode (the code that will represent the option group in your system).
- name
- integrationName
- product
- minimunQuantity
- maximunQuantity

notes: The product included in the optional group must exist.

```
    let product = {
      integrationCode: 'INT987'
    };

    let optionGroup = {
      integrationCode: 'VDE323',
      name: 'Extras',
      product: product,
      maximumQuantity: 10,
      minimumQuantity: 5
    };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
        assert(await apiClientToken.menu.optionGroup.delete(optionGroup));
    } catch (error) {
        assert.fail('Can not delete option group with token: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to delete the optional group. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it’s mandatory.

- With centralized credentials and restaurant Id

  ```
    try {
        assert(await apiClientToken.menu.optionGroup.delete(optionGroup,
  restaurantId));
    } catch (error) {
        assert.fail('Can not delete optional group with restaurant id: ' +
  error.message);
    }
  ```

- With centralized credentials and restaurant code

  ```
    try {
        assert(await apiClientToken.menu.optionGroup.delete(optionGroup,
  restaurantCode));
    } catch (error) {
        assert.fail('Can not delete optional group with restaurant code: ' +
  error.message);
    }
  ```

Get all option groups

To get all product's option groups you will need the following field:

- product (you need at least it's integrationCode)

notes: The product included in the optional group must exist.

```
function getOptionGroup() {
    let optionGroup = {
      integrationCode: 'ABC',
      name: 'Acompañamientos',
      product: { integrationCode: 'ABC' }
    };
    return optionGroup;
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let optionGroups = await api.menu.optionGroup.getAll(getOptionGroup());
      assert(optionGroups.length > 0);
    } catch (error) {
      assert.fail('Can not get all group options: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get all options group.

- With centralized credentials and restaurant Id

```
    try {
      let api = new ApiClient(
        getIntegrationCentralizedCredentials()
      );
       let optionGroups = await api.menu.optionGroup.getAll(getOptionGroup(),
  restaurantId);
       assert(optionGroups.length > 0);
    } catch (error) {
      assert.fail('Can not get all sections: ' + error.message);
    }
```

- With centralized credentials and restaurant code

```
    try {
      let api = new ApiClient(
        getIntegrationCentralizedCredentials()
      );
       let optionGroups = await api.menu.optionGroup.getAll(getOptionGroup(),
  restaurantCode);
       assert(optionGroups.length > 0);
    } catch (error) {
      assert.fail('Can not get all sections: ' + error.message);
    }
```

Note As response, you will receive a list of OptionGroups, each of them with the following attributes:

- integrationCode (the code that will represent the option group in your system).
- name
- integrationName
- minimunQuantity
- maximunQuantity

Get an Option Group by name or integrationCode

You will need the following fields to get an Option Group by name:

- name
- product (you need at least it's integrationCode).

If you want to get the Option Group by integrationCode:

- integrationCode (the code that will represent the option group in your system).
- product (you need at least it's integrationCode)

```
function getOptionGroup() {
    let optionGroup = {
      integrationCode: 'ABC',
      name: 'Acompañamientos',
      product: { integrationCode: 'ABC' }
    };
    return optionGroup;
  };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      assert (await api.menu.optionGroup.getByName(getOptionGroup()));
    } catch (error) {
      assert.fail('Can not get section: ' + error.message);
    }
```

```
    try {
      let response = await
api.menu.optionGroup.getByIntegrationCode(getOptionGroup());
      assert(response);
    } catch (error) {
      assert.fail('Can not get section: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get a option group by name or integration code.
You need to set as second parameter the restaurant id or restaurant code.

- Get by name or integration code with centralized credentials and restaurant Id

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let response = await api.menu.optionGroup.getByName(getOptionGroup(),
restaurantId);
    assert(response);
  } catch (error) {
    assert.fail('Can not get option group: ' + error.message);
  }
```

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let response = await
api.menu.optionGroup.getByIntegrationCode(getOptionGroup(), restaurantId);
    assert(response);
  } catch (error) {
    assert.fail('Can not get option group: ' + error.message);
  }
```

- Get by name or integration code with centralized credentials and restaurant code

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let response = await
api.menu.optionGroup.getByName(getOptionGroup(),restaurantCode);
    assert(response);
  } catch (error) {
    assert.fail('Can not get option group: ' + error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
    let api = new ApiClient(
      SDKTestUtil.getIntegrationCentralizedCredentials()
    );
    let response = await
api.menu.optionGroup.getByIntegrationCode(getOptionGroup(),restaurantCode);
    assert(response);
  } catch (error) {
    assert.fail('Can not get option group: ' + error.message);
  }
```

Create an optional

It's possible to create a new optional within an existing product. You have to follow the same rules as creating an optional group, meaning you have to create a product with it's integrationCode (mandatory) and if you have the productId. At the same time you have to create a optional group to contain the product, with the fields specified in the optional group section. Once you have that, you can create an optional, these are the fields you will need:

- integrationCode (mandatory)
- name (mandatory)
- optionGroup (mandatory: integrationCode)
- product (mandatory: integrationCode)
- price
- quantity
- enabled (true/false)
- modifiesPrice (true/false)
- requiresAgeCheck (true/false)

notes There's a few things you can do with the options' price, for example:

notes: When creating the optional the index will not be taken into account because it will be created in the last position.

Sent	Shown
Price 10	Adds 10 to the order value
Price 10, modifiesPrice true	Modifies the current order value to 10
No Price value	No changes to current optional price value
Price 0	optional price value changed to no cost

```
let product = {
  integrationCode: 'INT989'
};

let optionGroup = {
  integrationCode: 'TESTSKU',
  name: 'TESTSKU',
  product: product,
};

let option = {
  integrationCode: 'udf212',
  name: 'Lechuga Roja',
  optionGroup: optionGroup,
  price: 123,
  enabled: true,
  quantity: 2
};
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
    assert(await apiClientToken.menu.option.create(option));
} catch (error) {
    assert.fail('Can not create option with token: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to create the optional. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it's mandatory.

- With centralized credentials and restaurant Id

```
  try {
      assert(await apiClientToken.menu.option.create(option, restaurantId));
  } catch (error) {
      assert.fail('Can not create optional with restaurant id: ' +
error.message);
  }
```

- With centralized credentials and restaurant code

```
  try {
      assert(await apiClientToken.menu.option.create(option, restaurantCode));
  } catch (error) {
      assert.fail('Can not create optional with restaurant code: ' +
error.message);
  }
```

#####Update an optional To update the optional you need to follow the steps to create one, this means creating the product, optional group and following the restrictions mentioned before. The fields you will able to update are:

- integrationCode (mandatory)
- name (mandatory)
- optionGroup (mandatory: integrationCode)
- product (mandatory: integrationCode)
- price
- quantity
- enabled (true/false)
- modifiesPrice (true/false)
- requiresAgeCheck (true/false)

notes: The index will be taken into account if the section the product of the optional is located does not have a subsided or outstanding product.

notes: If the option is replicated in the menu (shares the same integrationCode), this operation will edit every option with the same integrationCode.

```
    let product = {
      integrationCode: 'INT989'
    };

    let optionGroup = {
      integrationCode: 'TESTSKU',
      name: 'TESTSKU',
      product: product,
    }

    let option = {
      integrationCode: 'MODIFYED123',
      integrationName: 'Lechuga',
      optionGroup: optionGroup,
      price: 500,
      enabled: true,
      quantity: 2
    };
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
  try {
      assert(await apiClientToken.menu.option.modify(option));
  } catch (error) {
      assert.fail('Can not modify option with token: ' + error.message);
  }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to update the optional. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it’s mandatory.

- With centralized credentials and restaurant Id

  ```
    try {
        assert(await apiClientToken.menu.option.modify(option, restaurantId));
    } catch (error) {
        assert.fail('Can not modify optional with restaurant id: ' +
  error.message);
    }
  ```

- With centralized credentials and restaurant code

  ```
    try {
        assert(await apiClientToken.menu.option.modify(option, restaurantCode));
    } catch (error) {
        assert.fail('Can not modify optional with restaurant code: ' +
  error.message);
    }
  ```

#####Delete an optional To delete the optional you need to follow the steps to create one, this means creating the product, optional group and following the restrictions mentioned before. The fields you will need are:

- integrationCode (mandatory)
- name (mandatory)
- optionGroup (mandatory: integrationCode)
- product (mandatory: integrationCode)
- price
- quantity
- enabled (true/false)
- modifiesPrice (true/false)
- requiresAgeCheck (true/false)

```
  let product = {
    integrationCode: 'INT989'
  };
```

```
let optionGroup = {
  integrationCode: 'TESTSKU',
  name: 'TESTSKU',
  product: product,

};

let option = {
  integrationCode: 'udf212',
  integrationName: 'Lechuga Roja',
  optionGroup: optionGroup,
  price: 123,
  enabled: true,
  quantity: 2
};
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don't need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
try {
    assert(await apiClientToken.menu.option.delete(option));
} catch (error) {
    assert.fail('Can not delete option with token: ' + error.message);
}
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to delete the optional. This will be added to the product object under partnerId or partnerIntegrationCode. Restaurant will contain either id or code, it's mandatory.

- With centralized credentials and restaurant Id

  ```
    try {
        assert(await apiClientToken.menu.option.delete(option, restaurantId));
    } catch (error) {
        assert.fail('Can not delete optional with restaurant id: ' +
  error.message);
    }
  ```

- With centralized credentials and restaurant code

  ```
    try {
        assert(await apiClientToken.menu.option.delete(option, restaurantCode));
    } catch (error) {
        assert.fail('Can not delete optional with restaurant code: ' +
  error.message);
    }
  ```

  Disable a product optional (Please use UPDATE for this functionality)
  Enable a product optional (Please use UPDATE for this functionality)
  Change product optional price (Please use UPDATE for this functionality)

Get all options

To get all option group options you will need the following fields:

- optionGroup (you need at least it's integrationCode) : the option group which the options belongs.
- product (you need at least it's integrationCode) : the product which the option group belongs.

notes: The product included in the optional group must exist.

```
function getOption() {
    let option = {
      optionGroup: {
        integrationCode: getIntegrationCode(),
        product: { integrationCode: getIntegrationCode() }
      }
    };
    return option;
  }
```

WITH POINT TO POINT CREDENTIALS

If you have a point to point integration you don’t need to add the restaurant id or the restaurant code because you will already have that information in the api credentials you used to login.

```
    try {
      let options = await api.menu.option.getAll(getOption());
      assert(options.length > 0);
    } catch (error) {
      assert.fail('Can not get all options: ' + error.message);
    }
```

WITH CENTRALIZED CREDENTIALS

If you have a centralized integration you can use either the restaurant id or the restaurant code to get all options of an optionGroup.

- With centralized credentials and restaurant Id

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
     let options = await api.menu.option.getAll(getOption(), restaurantId);
     assert(options.length > 0);
  } catch (error) {
    assert.fail('Can not get all options: ' + error.message);
  }
  ```

- With centralized credentials and restaurant code

  ```
  try {
    let api = new ApiClient(
      getIntegrationCentralizedCredentials()
    );
     let options = await api.menu.option.getAll(getOption(), restaurantCode);
     assert(options.length > 0);
  } catch (error) {
  ```

```
    assert.fail('Can not get all options: ' + error.message);
  }
```

Note As response, you will receive a list of OptionGroups, each of them with the following attributes:

- integrationCode (the code that will represent the option group in your system).
- name
- integrationName
- price
- requiresAgeCheck
- modifiesPrice
- quantity

Events operations

Some of the events operations are mandatory operations that must be implemented. Ignoring this operations will indeed cause the close of the restaurant or other kind of penalizations over PedidosYa platform.

Initialization event (Mandatory)

This event must be sent at the startup of the reception system. Should include all the possible information about the running device, for example: os version, application version, hardware id, etc.

WITH POINT TO POINT CREDENTIALS

```
let version = {};
version.os = 'Windows 10 x64';
version.app = '1.3.5';
api.event.initialization(version);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let version = {};
version.os = 'Windows 10 x64';
version.app = '1.3.5';
let restaurantId = restaurantId();
api.event.initialization(version, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let version = {};
version.os = 'Windows 10 x64';
version.app = '1.3.5';
let restaurantCode = restaurantCode();
api.event.initialization(version, restaurantCode);
```

Heartbeat event (Mandatory)

The heartbeat event must be sent every **two** minutes, this event proves that the reception system it's online.

WITH POINT TO POINT CREDENTIALS

```
let schedule = require('node-schedule');
schedule.scheduleJob('*/2 * * * *', async () => {
  try {
    api.event.heartBeat();
  } catch (error) {
    console.log(error);
  }
});
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let schedule = require('node-schedule');
schedule.scheduleJob('*/2 * * * *', async () => {
  try {
    let restaurantId = restaurandId();
    api.event.heartBeat(restaurantId);
  } catch (error) {
    console.log(error);
  }
});
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let schedule = require('node-schedule');
schedule.scheduleJob('*/2 * * * *', async () => {
  try {
    let restaurantCode = restaurandCode();
    api.event.heartBeat(restaurantCode);
  } catch (error) {
    console.log(error);
  }
});
```

Reception event (Mandatory)

This event must be called when an order arrives to the reception system.

WITH POINT TO POINT CREDENTIALS

```
let orderId = orderReceivedId();
api.event.reception(orderId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let orderId = orderReceivedId();
let restaurantId = restaurantId();
api.event.reception(orderId, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let orderId = orderReceivedId();
let restaurantCode = restaurantCode();
api.event.reception(orderId, restaurantCode);
```

Acknowledgement event (Mandatory)

The acknowledgement event must be sent when the restaurant **sees** the order and not before.

WITH POINT TO POINT CREDENTIALS

```
let orderId = orderReceivedId();
api.event.acknowledgement(orderId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let orderId = orderReceivedId();
let restaurantId = restaurantId();
api.event.acknowledgement(orderId, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let orderId = orderReceivedId();
let restaurantCode = restaurantCode();
api.event.acknowledgement(orderId, restaurantCode);
```

State change event

Every time that you want to change the state of an order you should register a event of this kind.

WITH POINT TO POINT CREDENTIALS

```
let orderId = orderReceivedId();
api.event.stateChange(orderId, OrderState.CONFIRMED);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let orderId = orderReceivedId();
let restaurantId = restaurantId();
api.event.stateChange(orderId, OrderState.CONFIRMED, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let orderId = orderReceivedId();
let restaurantCode = restaurantCode();
api.event.stateChange(orderId, OrderState.CONFIRMED, restaurantCode);
```

Warning event

This event represents that the reception system is in a warning state, for example: low battery, lack of paper, etc. You must provide the warning internal identification code and a description.

WITH POINT TO POINT CREDENTIALS

```
let warningCode = 'LOW_BAT';
let warningDescription = 'Low battery. Please plug in the device';
api.event.warning(warningCode, warningDescription);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let warningCode = 'LOW_BAT';
let warningDescription = 'Low battery. Please plug in the device';
let restaurantId = restaurantId();
api.event.warning(warningCode, warningDescription, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let warningCode = 'LOW_BAT';
let warningDescription = 'Low battery. Please plug in the device';
let restaurantCode = restaurantCode();
api.event.warning(warningCode, warningDescription, restaurantCode);
```

Error event

This event represents a error, for example: missing product code, can't confirm order, error processing order. You must provide the error internal identification code and a description.

WITH POINT TO POINT CREDENTIALS

```
let errorCode = 'MISSING_CODE';
let errorDescription = 'Missing code for product with description Burger XL';
api.event.error(errorCode, errorDescription);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT ID

```
let errorCode = 'MISSING_CODE';
let errorDescription = 'Missing code for product with description Burger XL';
let restaurantId = restaurantId();
api.event.error(errorCode, errorDescription, restaurantId);
```

WITH CENTRALIZED CREDENTIALS AND RESTAURANT EXTERNAL CODE

```
let errorCode = 'MISSING_CODE';
let errorDescription = 'Missing code for product with description Burger XL';
let restaurantCode = restaurantCode();
api.event.error(errorCode, errorDescription, restaurantCode);
```

Models

REQUEST MODELS

In this chapter is the description of the different models you will be seen in different operation request.

Product

All the product data.

- **id** (int): The product identification number.
- **name** (string): The product name. For example: 'Pizza', 'Burger'.
- **description** (string): The product description.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.
- **section** (Section): The product section.
- **image** (string): The product image url.
- **price** (double): The product section.

RESPONSE MODELS

In this chapter is the description of the different models you will be seen in different operation responses.

Address

All the needed information about the user address. The address is an attribute of the order.

- **complement** (string): The user address complement. For example: Yellow house.
- **phone** (string): The user address phone.
- **area** (string): The user area, a.k.a: neighborhood.
- **corner** (string): The user address street corner.

- **description** (string): The address formatted: street, corner, number, etc. For example: 'Plaza Independencia 743 esquina Ciudadela'.
- **street** (string): The user address street.
- **zipCode** (string): Returns the user address zip code. This fields depends on the country, for example in Uruguay doesn't make much sense but in Brazil references to the user CEP.
- **notes** (string): The address notes things such as 'the blue house', 'the ring doesn't work', etc.
- **doorNumber** (string): The user address door number.
- **coordinates** (string): The user address coordinates.
- **city** (string): The user city.

Attachment

All the information related to an attachment.

- **url** (string): The attachment url

DeliveryTime

All the needed information about the possible delivery times for the orders.

- **id** (int): The delivery time identification number.
- **name** (string): The delivery time identification name, ex: Entre30Y45.
- **description** (string): The delivery time description, ex: Entre 30' y 45'.

Details

All the needed information about the order details. It contains a list of the order products with their options and extra info.

- **product** (Product): The associated product.
- **optionGroups** (array of OptionGroup): The product options. For example: 'Ketchup', 'Bacon'.
- **quantity** (int): The amount of products.
- **unitPrice** (double): The product unit price.
- **subtotal** (double): The product subtotal.
- **notes**: The notes included for the product. For example: 'The burger without tomato please.'.
- **discount** (double): The discount amount of the product if apply.
- **total** (double): The total amount of the product with discounts and price modifications.

Discount

All the information related to a discount.

- **amount** (double): Net amount.
- **priority** (int): Priority of the discount in relation with other discounts in the list.
- **value** (double): Original value of the discount.
- **type** (DiscountType): The discount type of the discount.
- **notes** (string): Discount Notes.

DiscountType

Type of the payment discount in the payment attribute of the order.

- **PERCENTAGE** (string): Type of discount when it's specified by percentage value.
- **VALUE** (string): Type of discount when it's specified by an amount of money.

Option

All the needed information about the product option.

- **id** (int): The option identification number.

- **name** (string): The option name.
- **index** (int): The option index. For sorting purposes.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.
- **quantity** (int): The quantity of selected options.
- **amount** (double): The amount of the option.
- **modifiesPrice** (bool): true if the option modifies the price of the product.

OptionGroup

All the product options.

- **name** (string): The option group name.
- **index** (int): The option group index. For sorting purposes.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.
- **options** (array of Option): The options associated to the group.

Order

All the order data. The order details, client information, payment method, etc.

- **application** (string): Return the application type that the order has been created. It's one of the following: WEB, CALL, IPHONE, ANDROID, WIDGET, WINDOWS_PHONE, WEB_MOBILE, IPAD.
- **payment** (Payment): The payment method of the order.
- **pickup** (bool): true if the order is for pickup, false if is for delivery.
- **responseDate** (DateTime): The order response date.
- **dispatchDate** (DateTime): The order dispatch date
- **portal** (Portal): The portal of the order that belongs to.
- **registeredDate** (DateTime): The order registration date.
- **state** (OrderState): The order state.
- **code** (string): The order identification code.
- **restaurant** (Restaurant): The order restaurant.
- **express** (bool): true if the order is an express order, false if is for delivery.
- **id** (int): The order identification number
- **details** (array of Detail): The order details data.
- **discounts** (array of Discount): The list of discounts applied.
- **address** (Address): The user address.
- **deliveryDate**: Returns the delivery date of the order.
- **notes** (string): The order notes.
- **whiteLabel** (string): The white label of the order that belongs to.
- **user** (User): The user who made the order.
- **attachments** (array of Attachment): The list of attachments associated to the order.

OrderState

- **PENDING** (string): State of the order when is new and ready to be answered.
- **CONFIRMED** (string): State of the order when is confirmed.
- **REJECTED** (string): State of the order when is rejected.

Payment

All payment information of the order.

- **total** (double): The total price of the order.
- **subtotal** (double): The sum of details totals plus shipping.
- **tax** (double): The total taxes amount of the order.
- **online** (bool): true if it's online payment.

- **id** (int): The payment method identification number.
- **shipping** (double): The shipping cost of the order.
- **paymentAmount** (double): The user payment amount.
- **amountNoDiscount** (double): The amount of the order without discount.
- **currencySymbol** (string): The currency symbol of the payment method .
- **shippingNoDiscount** (double): The shipping cost of the order without discount.
- **method** (string): The payment method description: 'Cash', 'Credit card', etc.
- **notes** (string): The payment notes, for example: '$100 with credit card and $17 with cash'.

Portal

All the information about the order platform.

- **id** (int): The portal identification number.
- **name** (string): The portal name. For example: 'Pedidos Ya', 'Pizza Pizza'

Product

All the product data.

- **id** (int): The product identification number.
- **name** (string): The product name. For example: 'Pizza', 'Burger'.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.
- **section** (Section): The product section.

RejectMessage

All information about possible rejection messages.

- **id** (int): The reject message identification number.
- **descriptionES** (string): The reject message description in Spanish.
- **descriptionPT** (string): The reject message description in Portuguese.
- **forLogistics** (bool): Whether the reject message must be used for logistic orders or not.
- **forPickup** (bool): Whether the reject message must be used for pickup orders or not.

Restaurant

All the restaurant information of an order.

- **id** (int): The restaurant identification number.
- **name** (string): The restaurant name.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.

Section

All the information about the section of a product.

- **index** (int): The section index. For sorting purposes.
- **name** (string): The section name. For example: 'Pizzas', 'Beverages'.
- **integrationCode** (string): The external integration code.
- **integrationName** (string): The external integration name.

User

All the user data. The user email, full name, etc.

- **platform** (string): The platform that the user belongs to.
- **lastName** (string): The user last name.
- **email** (string): The user email.

- **isNew** (bool): true if the user is new.
- **name** (string): The user name.
- **identityCard** (string): The user identity card. For example, the Uruguayan CI number, the Brazilian CNPF or CNPJ number, etc.
- **type** (string): The user type.
- **orderCount** (int): The amount of orders that the user has on the specified platform.

Tracking

All the needed information about the order's tracking

- **state** (TrackingState): The tracking's state.
- **driver** (Driver): Information about the driver responsible to deliver the order,
- **pickupDate** (DateTime): The date the driver will pickup the order from store.
- **estimatedDeliveryDate** (DateTime): The date the driver will deliver the order to the client,

TrackingState

- **FAILURE** (string): State of the tracking when it failed.
- **REQUESTING_DRIVER** (string): State of the tracking when a driver is being requested.
- **TRANSMITTING** (string): State of the tracking when the order is being transmitted.
- **TRANSMITTED** (string): State of the tracking when the order is finally transmitted.
- **PREPARING** (string): State of the tracking when the order is being prepared.
- **DELIVERING** (string): State of the tracking when the order is being delivered.
- **DELIVERED** (string): State of the tracking when the order is finally delivered.
- **CLOSED** (string): State of the tracking when the order is closed and no live tracking happens.

Driver

- **name** (string): The driver's name.
- **location** (Location): The driver's location.

Location

- **lat** (double): The location's latitude.
- **lng** (double): The location's longitude.