SDK Methods

The following methods are available to use in the SDK. All methods are part of the personalization class.

For parameters that require event names, refer to SDK Events and Action Types for syntax.

report

Reports an event to Monetate. This allows data to later be used for decisions within Monetate. Use this method to report events.

```
report(context: <ContextEnum>, event: <MEvent?>)
```

Parameters:

- `context` is name of the event. (Required)
- `event` is the event data. (Required)

Example code:

```
Personalization.shared.report(context: .RecClicks, event: RecClicks(recClicks: ["rt.1.xxx", "rt.1.yyy"
```

addEvent

Reports an event to Monetate. Use this method to report events that you intend to use to trigger experiences. You can use multiple calls of this method to report multiple events for experiences that require multiple conditions.

This method is used to report events that are used by the `getActionsData` method to determine an experience. If you use this method, you must use `getActionsData` to trigger the experience that satisfies the reported events.

```
addEvent(context: <ContextEnum>, event: <MEvent?>)
```

Parameters:

- `context` is name of the event. (Required)
- `event` is the event data. (Required)

Example code:

```
Personalization.shared.addEvent(context: .ScreenSize, event: ScreenSize(height: 1000, width: 400)

Personalization.shared.addEvent(context: .IpAddress, event: IPAddress(ipAddress: "192.168.1.52"))

Personalization.shared.addEvent(context: .PageView, event: PageView(pageType: "PDP", path: "n/a
```

getActionsData

Request an experience decision from Monetate based off the action type. You can specify multiple action types in an array to get multiple responses.

The experience decision depends on event data reported using `addEvent` calls. Use `addEvent` to add report all of the relevant events before you use this method to request a decision.

```
getActionsData(requestId: <String>, arrActionTypes: <[ActionTypeEnum]>)
```

Parameters:

- `requestID` is the request ID for the API.
- `actionType` is the type of action you want to request. You can specify multiple actions in an array to handle. (Required)

Example code specifying a single action:

```
Personalization.shared.getActionsData(
  requestId: "123456",
  arrActionTypes: [.OmniChannelRecommendation]
).on { res in
  if res.status == 200 {
    self.handleRecommendations(res: res)
  } else {
  }
}
```

Example code specifying multiple actions:

```
Personalization.shared.getActionsData(
  requestId: "123456",
  arrActionTypes: [.OmniChannelJson, .OmniChannelRecommendation]
).on { res in
  if res.status == 200 {
    self.handleRecommendations(res: res)
  } else {
  }
}
```

getActions

Reports an event and immediately requests a decision from Monetate. Use this method if an experience you want to trigger requires a single event. This method returns a JSON object that includes the response data.

The response data can then be used in your application. For example, you can use this method to obtain data to display as a banner on a page.

```
getActions(context: <ContextEnum>, requestId: <String>, arrActionTypes: <[ActionTypeEnum]>,
```

Parameters:

- `context` is name of the event. (Optional)
- `requestID` is the request ID for the API.
- `actionTypes` is the type of action you want to request. You can specify multiple actions in an array to handle. (Required)
- `event` is the data associated with the event. (Optional)

Example code:

```
import MarqueeLabel
import UIKit
import monetate_ios_sdk

class CategoryViewController: UIViewController {

 @IBOutlet weak var bottomLabel: MarqueeLabel!
 @IBOutlet weak var bottomView: UIView!
 @IBOutlet weak var constraintHeightBottomView: NSLayoutConstraint!

 override func viewDidLoad() {
  super.viewDidLoad()
  self.bottomView.isHidden = true
  self.constraintHeightBottomView.constant = 0
  bottomLabel.text = ""
 }

 override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)
  pageEvent()
 }

 func pageEvent() {
  Personalization.shared.getActions(
   context: .PageView, requestId: "test_request_id", arrActionTypes: [.OmniChannelJson],
   event: PageView(
    pageType: "Homepage", path: "n/a", url: "n/a", categories: [], breadcrumbs: [])
  ).on { (res) in
   if res.status == 200 {
    let data = JSON(res.data)
    print(data)
    self.handleAction(res: res)
   } else {
   }
  }
```

```
 }

fileprivate func handleAction(res: APIResponse) {
  let data = JSON(res.data)
  for item in data["data"]["responses"].arrayValue {
    bottomLabel.animationCurve = .linear
    bottomLabel.speed = .duration(5)
    bottomLabel.fadeLength = 15.0

    if item["requestId"].string == res.requestId {
      for oneaction in item["actions"].arrayValue {
        let component = oneaction["component"].string ?? ""
        if component.lowercased() == "footer" {
          if let json = oneaction["json"].dictionary {
            print("final dict \(json)")
            if let text = json["text"]?.string {
              self.bottomView.isHidden = false
              self.constraintHeightBottomView.constant = 60
              bottomLabel.text = text
              if let fontStyle = json["style"]?.string, let fontSize = json["fontSize"]?.double {
                if fontStyle == "bold" {
                  bottomLabel.font = UIFont(name: "Roboto-Bold", size: fontSize)
                } else if fontStyle == "normal" {
                  bottomLabel.font = UIFont(name: "Roboto-Regular", size: fontSize)
                } else if fontStyle == "italic" {
                  bottomLabel.font = UIFont(name: "Roboto-Italic", size: fontSize)
                }
              }
              if let color = json["color"]?.string {
                bottomLabel.textColor = MiscClass.hexStringToUIColor(hex: color)
              }
              if let backgroundColor = json["background-color"]?.string {
                bottomView.backgroundColor = MiscClass.hexStringToUIColor(hex: backgroundColor)
                return
              }
            }
          }
        }
      }
    }
  }
}
```

flush

Immediately sends all event reports that are currently queued. Use this method if you want to report an event immediately. This method returns a response of success or failure.

This method might throw the following exceptions that you must handle:

- `InterruptedException`
- `ExecutionException`
- `TimeoutException`

```
flush()
```

Example code:

```
Personalization.shared.report(context: .RecClicks, event: RecClicks(recClicks: ["rt.1.xxx", "rt.1.yyy"
Personalization.shared.report(context: .RecImpressions, event: RecImpressions(recImpressions: ["r
Personalization.shared.flush()
```

setCustomerId

Updates the `customerId` within the User object.

```
setCustomerId(customerId)
```

Parameters:

- `customerId` is a string containing the customer ID. (Required)

Example code:

```
Personalization.shared.setCustomerId(customerId:"155468")
```