Create an Omnichannel Recommendations Action

Omnichannel recommendations is an integrated recommendations experience that is consistent across all platforms that you connect to Monetate. Whether a customer is viewing your storefront through a mobile app or your site, Omnichannel recommendations ensure a consistent experience.

You can set up a handler for an Omnichannel recommendations action by using two methods. The `addEvents` method defines the events that can trigger the action. The `getActionsData` method is then used as the trigger and requests the decision based on the defined events. `getActionsData` then returns a JSON object containing recommendations data that you can then handle in code.

Prerequisites

You must first create an Omnichannel experience within Monetate for the methods to reference. Refer to Configure an Omnichannel Recommendations Action for instructions.

Make note of the WHO settings, as these correspond to events your code listens for. The example experience in this article uses the following WHO settings:

- IP address is 1.0.0.2
- Screen height is at least 500 pixels and screen width is at least 300 pixels

The example code in this article fulfills these conditions and will trigger the Omnichannel experience.

addEvent

This method records a local event in the defined context.

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

Parameters:

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

You can use this method multiple times to add all the necessary events for an experience you might want to trigger. The example code uses multiple method calls to fulfill the experience requirements:

Code Example

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

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

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

getActionsData

This method sends the defined events to Monetate to trigger an experience. If the events fulfill the WHO settings of an experience, then that experience is triggered. A JSON object containing the experience response is then returned.

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

Parameters:

- `requestID` is the request ID for the API.
- `includeReporting` indicates whether the response will have impression reporting data.
- `arrActionTypes` is the types of action you want to request. You can specify multiple actions in an array to handle.

Code Example

```
objPersonalization.getActionsData(requestId: "123456", includeReporting: false, arrActionTypes:["n
 if res.status==200 {
   self.handleRecommendations(res: res)
 } else {
 }
}
}
```

Full Code Example

Complete code example blocks are listed below.

```
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!
 final var objPersonalization = Personalization(
   account: Account(
```

```
    instance: "p", domain: "localhost.org", name: "a-701b337c", shortname: "localhost"),
  user: User(deviceId: "62bd2e2d-213d-463f-83bb-12c0b2530a14"))

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() {
  objPersonalization.getActions(
    context: .PageView, requestId: "test_request_id", includeReporting: false,
    arrActionTypes: ["monetate:action: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
                }
              }
            }
          }
        }
      }
    }
  }
}
```