Create an Image-Based Badging Action

The product badging feature allows clients to apply some treatment to specific items on product listing pages and product detail pages to draw site visitors' attention to those items to foster engagement. Product badging highlights key attributes of items that might not otherwise jump out to visitors navigating products on the site.

A product badge frequently appears as a digital promotional sticker displayed atop the product image to indicate that the item is a bestseller or a top-rated product. These labels serve as customer endorsements, allowing shoppers to more quickly locate popular products on a page.

You can set up a handler for Omnichannel badging actions 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 experience based on the defined events. `getActionsData` then returns a JSON object containing badging image data that you can then handle in code.

Prerequisites

You must first create an Omnichannel badging action within Monetate for the methods to reference. Refer to Configure an Omnichannel Product Badging Action for instructions.

This action uses Product Thumbnail View as an event listener. To accomplish this, you must set **Thumbnail's product ID =** as one of the conditions. Make note of this product ID so that you can pass the appropriate ID in code.

Optionally, you can add other conditions to this action. If you do, make note of those so that you can pass those as relevant events.

addEvent

This method records a local event.

```
public void addEvent(String context, Object event)
```

Parameters:

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

For an Omnichannel Badging action, use the `ContextProductThumbnailView` event as your context. Pass a list of product IDs that you want to set up a badge for as your events data.

```
// Events/context
ProductThumbnailView productThumbnailView = new ProductThumbnailView();

// addEvent
personalization.addEvent(EventTypes.ContextProductThumbnailView, productThumbnailView);
```

If you added other conditions to your Omnichannel Badging action, use additional calls of this method to handle them with the appropriate events and data.

getActionsData

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

```
public String getActionsData(String[] actionTypes)
```

Parameters:

- `actionTypes` is the type of action you want to request. You can specify one action or multiple actions in an array to handle. (Required)

Use `OmniChannelImageBadging` as the action type for this method.

```
responseData = personalization.getActionsData(ActionTypes.OmniChannelImageBadging);
```

Full Code Example

Complete code example blocks are listed below.

```
Personalization personalization = new personalization(user, account);

// Events/Context data
ProductThumbnailView productThumbnailView = new ProductThumbnailView();
productThumbnailView.setProducts(new String[]
{
  "SLBC",
  "99"
});

// addEvent
personalization.addEvent(EventTypes.ContextProductThumbnailView, productThumbnailView);

// getActionsData
String responseData;
new Thread(new Runnable()
{

  @Override
  public void run()
  {

    // Gets the responseData using getActionsData
```

```
  responseData = personalization.getActionsData(ActionTypes.OmniChannelImageBadging);

  // Once you receive the responseData from getActionsData, use the Handler and parse the requi
  new Handler(Looper.getMainLooper()).post(new Runnable()
  {
    @Override
    public void run()
    {

      // Parse the received responseData, get the requiredData from it and use it accordingly
      try
      {
        // Example: textView.setText(requiredData);
      }
      catch (Exception ex)
      {
        // Catch any exception that occured while parsing
        ex.printStackTrace();
      }

    }
  });
 }
}).start();
```

You must handle the getActions method in a Thread or AsyncTask. To update the main thread, you can use either Handler or runOnUiThread. The code example above uses Thread and Handler to update the main thread and change the UI.

Response Data

An example of the JSON data returned from getActionsData is below. Handle this response in your code accordingly to render the data in your UI as you see fit.

```
{
 "actions": [
   {
    "image_content": {
     "version": 2,
     "clickzones": [],
     "contentId": 775504,
     "top": 0,
     "title": "Test-Badge",
     "href": "https://marketer.monetate.net",
     "iwidth": 40,
     "alt": "Alt-Test-Badge",
     "iheight": 40,
     "ref": "https://sb.monetate.net/img/1/1094/4631519.jpg",
     "left": 180
    },
    "actionType": "monetate:action:OmniChannelImageBadging",
    "actionId": 4959427,
    "application_data": {
     "test": "React-native SDK"
    },
    "positioning_hint": "center",
    "pids": [
     "99",
     "SLBC"
    ]
   }
 ],
 "requestId": "81.2272963332645"
}
```