Unbxd Android SDK

Prerequisites

Before you get started with the integration, you would need to:

1. Set up your Site Search Dashboard.
2. Upload your Product Feed

If you have not yet registered, you can sign-up with Unbxd Site Search here

Configure

1. Add Unbxd Android SDK to your project (It will automatically include Volley library).
2. Configure your Keys, by calling UnbxdGlobal.configure() method as follows:
   UnbxdGlobal.configure(<ACTIVITY_NAME>,<SITE_KEY>, <API_KEY>, <SECRET_KEY>);

- **SITE_KEY**: Your SITE KEY from the overview section
- **API_KEY**: Your API from Profile Settings section.
- **SECRET_KEY**: Your SECRET KEY from Profile Settings section.

1. Integrating Site Search

Response

Integration of site search methods defined below gives out a response which will be handled in following way:
Success Response: will be of **JSONObject** type.
Error Response: will be of **VolleyError** type.

```
UnbxdSearch.setOnAPICallbackListener(new APICallBackListener.OnAPICallbackListener() {
            @Override
            public void onSuccessResponse(JSONObject jsonObject) {
              // success response logic over here
            }

            @Override
            public void onErrorResponse(VolleyError volleyError) {
              // error response logic over here
            }
          });
```

Search functions:

The following code will make a search call -:

1.1: Simple Search

Syntax: **UnbxdSearch**.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);
Example: UnbxdSearch.searchFor(MainActivity.this, “shirts”);

1.2: Search with pagination

Syntax:
UnbxdSearch.**paginationParams**(<Row Pagination>, <Start Pagination>);
UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

Row Pagination: specify the **number of products** displayed in each page.
Start Pagination: specifies the **index** of the first product in a search listing page.

Example:
UnbxdSearch.paginationParams(2, 10);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

1.3: Filtering result

- **Text Filter :**
  Syntax:
  UnbxdSearch.filterParams(<Field Name>, <Field Value>);
  UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

  Field Name: specify the **name of field** on which filter is to be applied.
  Field Value: specify the **field value** on which you want to apply filter.

  Example:
  UnbxdSearch.filterParams(“color”, “blue”);
  UnbxdSearch.searchFor(MainActivity.this, “shirts”);

- **Range Filter:**
  Syntax:
  UnbxdSearch.filterParams(<Field Name>, <Field Value>, <Lower Limit>, <Upper Limit>);
  UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

  Field Name: specify the **name of field** on which filter is to be applied.
  Field Value: specify the **field value** on which you want to apply filter.

Lower Limit : specify the lower limit of the range.
Upper Limit: specify the upper limit of the range.

Example:
UnbxdSearch.filterParams(“color”, “blue”, 0, 100);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

- **Multiple Filter:**
  Syntax:
  UnbxdSearch.filterParams(<Hash_Map filter>);
  UnbxdSearch.searchFor(<ACTIVITY_NAME>,<QUERY TEXT>);

  Hash_Map filter: specify the **key-value** pair for the **fieldName-fieldValue** on which filter is to be applied.

  Example:
  Map<String, String> filterMulti = new HashMap<String, String>();
  filterMulti.put(**"category"**, **"men"**);
  filterMulti.put(**"color"**, **"Navy"**);
  UnbxdSearch.filterParams(filterMulti);
  UnbxdSearch.searchFor(MainActivity.this, “shirts”);

1.4: Facet Response
By default Search API returns the facet as part of the response.

Syntax:
UnbxdSearch.facetParams(<Facet Status>);
UnbxdSearch.searchFor(<ACTIVITY_NAME>,<QUERY TEXT>);

Example:
UnbxdSearch.facetParams(false);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

1.5: Multi-Selecting Facet Response
By default multi-select facet if disabled.

Syntax:
UnbxdSearch.facetMultiParams(<Facet Status>);
UnbxdSearch.searchFor(<ACTIVITY_NAME>,<QUERY TEXT>);

Example:
UnbxdSearch.facetMultiParams(true);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

1.6: Sorting
You can sort your result.

Syntax:
UnbxdSearch.sorting(<Field Name>, <Order>);
UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

Field Name: specify the **field name** that is to be sorted.
Order: specify the **order** of sorting i.e. **asc** or **desc**

Example:
UnbxdSearch.sorting(“price”, “asc”);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

Multiple Sorting
Syntax:
UnbxdSearch.sorting(<Hash_Map params>);
UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

Hash_Map filter: specify the **key-value** pair for the **fieldName-sorting_order** on which sorting is to be applied.

Example:
Map<String, String> sortingMulti = new HashMap<String, String>();
sortingMulti.put(**"category"**, **"asc"**);
sortingMulti.put(**"price"**, **"desc"**);
UnbxdSearch.sorting(sortingMulti);
UnbxdSearch.searchFor(MainActivity.this, “sportcoat”);

1(e): Bucketed Search
Syntax:
UnbxdSearch.bucketSearch(<Bucket Field> , <Bucket Limit>, <Bucket Offset>, <rows>, <Bucket sort>);
UnbxdSearch.**searchFor**(<ACTIVITY_NAME>,<QUERY TEXT>);

Bucket Field: The name of the field to be bucketed.
Bucket Limit: Determines the maximum number of buckets. By default, first 10 bucket will be displayed.
Bucket Offset: To paginate to the next 5 products of each group.
Rows: Determines the number of products shown in each group.
Bucket Sort: Determines the sorting order.

Example:
UnbxdSearch.bucketSearch(“price” , 4, 3,100, “desc”);
UnbxdSearch.searchFor(MainActivity.this, “shirts”);

2. Integrating Autosuggest Response

Response for AutoSuggest will be in the same format as for the Site Search which is described over here.

Syntax:
UnbxdAutosuggest.autoSuggest(<Activity Name>, <Query>, <inFields.count>, <popularProducts.count>, <keywordSuggestions.count>, <topQueries.count>);

Query: specify text on which autosuggest needs to be triggered (Mandatory).
inFields.count: Defines number of autosuggest results with doctype IN_FIELD. The default value is 2.
popularProducts.count: Defines number of autosuggest results with doctype POPULAR_PRODUCTS. The default value is 3.
keywordSuggestions.count: Defines number of autosuggest results with doctype KEYWORD_SUGGESTION. The default value is 2.
topQueries.count: Defines number of autosuggest results with doctype TOP_QUERIES. The default value is 2.

Example:
UnbxdAutosuggest.autoSuggest(MainActivity.this, “shi”, 3, 4, 2, 3);

3. Integrating Analytics

The Unbxd Analytics helps to make request calls to our servers to track each visitor event in your application. In order to get started, you must create an instance of the class.

UnbxdAnalytics analytics = new UnbxdAnalytics(<Activity Name>);

Parameters :

- **pid:** Unique ID of the product
- **query:** The search query
- **qty:** Quantity of ordered product
- **price:** Price of the ordered product

Response :

Integration of site search methods defined below gives out a response which will be handled in following way:

Success Response: will be of **String** type.

Error Response: will be of **VolleyError** type.

```
UnbxdAnalytics.setOnAPICallbackListener(new
APICallBackListener.OnAPICallbackStringListener() {
            @Override
            public void onSuccessResponseString(String response) {
               // Response logic over here
            }

            @Override
            public void onErrorResponse(VolleyError volleyError) {
               // Error Logic over here
            }

            @Override
            public void onStatusCode(int statusCode) {
               // Status Code
            }
        });
```

Tracking Events

Once you have successfully created an instance of the UnbxdAnalytics, we can now start tracking event as:

3.1 Search

Every time a visitor searches for a query, the below method along with its parameter values needs to be called:

Syntax:

```
Map<String,String> params = new HashMap<String, String>();
params.put(“query”, <query text>);
params.put(“url”, <url>);
params.put(“referrer”, <referrer url>);
params.put(“ver”, <ver>);
analytics.track("search", <Hash Map params>);
```

3.2 Click

Every time a visitor clicks on a product, below method along with its parameter values needs to be called:

Syntax:

```
Map<String,String> params = new HashMap<String, String>();
params.put(“pid”, <product id>);
params.put(“url”, <url>);
params.put(“referrer”, <referrer url>);
params.put(“ver”, <ver>);
analytics.track("click", <Hash Map params>);
```

3.3 Add to Cart

Every time a visitor adds a product in the cart, below method along with its parameter values needs to be called:

Syntax:

```
Map<String,String> params = new HashMap<String, String>();
params.put(“pid”, <product id>);
params.put(“url”, <url>);
params.put(“referrer”, <referrer url>);
params.put(“ver”, <ver>);
analytics.track("cart", <Hash Map params>);
```

3.4 Order

Every time a visitor orders a product and completes the payment procedure, below method along with its parameter values needs to be called:

Syntax:

```
Map<String,String> params = new HashMap<String, String>();
params.put(“pid”, <product id>);
params.put(“price”, <price>);
params.put(“qty”, <quantity>);
params.put(“url”, <url>);
params.put(“referrer”, <referrer url>);
params.put(“ver”, <ver>);
analytics.track("order", <Hash Map params>);
```

3.5 Visitor Event

Every time a visitor visits , below method along with its parameter values needs to be called:

Syntax:

```
Map<String,String> params = new HashMap<String, String>();
params.put(“url”, <url>);
params.put(“referrer”, <referrer url>);
params.put(“ver”, <ver>);
analytics.track("visitor", <Hash Map params>);
```