



Introduction

UnbxSDK implements support for making network calls to Unbx platform and lets to easily configure and integrate Unbx Site Search in your eCommerce application.

The following features are currently supported with UnbxSDK,

1. **Site search** : The Search API lets you interact with the Unbx platform and implement all search related functionality with ease.
2. **Auto suggest** : Unbx supports autocompletion of search queries and showcasing products relevant to query as they type.
3. **Browse** : The Browse API lets you interact with the Unbx platform and implement all category related functionality with ease.
4. **Analytics** : SDK provides API for visitor events such as product clicks, products added to cart, orders, etc.
5. **Recommendations** : Unbx supports product recommendations with help of recorded events.

Prerequisites

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

- Set up your Site Search Dashboard and obtain API key, Site key.
- Upload your Product Feed.

Dependency

Alamofire 4.7 or above - Alamofire is an HTTP networking library written in Swift

Supported Platform

UnbxSDK is a dynamic framework programmed using swift 4. This can be integrated with iOS application with version 9.0 and above.

Installation

Cocoapods is a dependency manager for Cocoa projects. You can install it with below command:

```
$ gem install cocoa pods
```

To integrate Unbxdsdk into your Xcode project using Cocoapods, specify it in your Podfile:

```
source '<source_url>'
platform :ios, '9.0'
use_frameworks!

target '<your target name>' do
  pod 'Unbxdsdk'
end
```

Run the below command:

```
$ pod install
```

Initialising SDK

- Import Unbxdsdk framework as shown below:

```
import Unbxdsdk
```

- Unbxdsdk is initialised with API key and Site key.

```
let sdkConfiguration = UbConfiguration.init(siteKey: "your_site_key", apiKey: "your_api_key")
```

```
let error = UbClient.sharedUbClient.initWithConfiguration(configuration: sdkConfiguration)
```

All the SDK methods are invoked on shared instance of UbClient.

Using Search Methods

Search methods performs search with key and few other parameters. Search key is the mandatory argument and rest is applied for different criteria.

Search method signature:

```
func searchWithQuery(query:SearchQuery, variantOption: VariantOption? = nil,  
returnFields:Array<String>? = nil, facet: Facet = .None, filter: FilterAbstract? = nil,  
categoryFilterPath: CategoryFilterAbstract? = nil, multipleFilter: MultipleFilterAbstract?  
= nil, fieldNamesWithSortOrder: Array<FieldSortOrder>? = nil, completion: @escaping  
(_ response:Any?,_ error:Error?) -> Void) {  
    // Handle response or request  
}
```

- * Keywords marked in bold are different arguments for Search method.
- * 'query' of type 'SearchQuery' is mandatory argument and rest are optional which are used as needed.

Lets see how these arguments can be composed and passed in searchWithQuery() method invocation.

A. SEARCHQUERY

SearchQuery consists of searchKey parameter with few other parameters.

Invoking Search method with SearchQuery

```
var searchQuery = SearchQuery.init(withSearchKey: "Shirt")  
  
//Optional parameters  
searchQuery.rowsCount = 20  
searchQuery.pageIndex =  
searchQuery.responseFormat = ResponseFormat.JSON  
searchQuery.spellCheck = true
```

```

searchQuery.analytics = true
searchQuery.showStatsForField = "price"

client?.searchWithQuery(query: searchQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})

```

B. VARIANTS

Variants parameter enables or disables variants in the API response. It can take two values: “true” or “false”. Default value is “false”.

```

let variant = VariantOption.init(withVariantStatus: true)
client?.searchWithQuery(query: searchQuery, variantOption: variant, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

If you want to get more than one variants in the API response, you can use ‘variantCount’ parameter. It can have any numerical value (eg, 1,2,3,etc) or “.max” (to get all the variants).

With Variant count value

```

let variant = VariantOption.init(withVariantStatus: true, variantCount: 2)
client?.searchWithQuery(query: searchQuery, variantOption: variant, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

With all Variants

```

let variant = VariantOption.init(withVariantStatus: true, variantCount: .max)
client?.searchWithQuery(query: searchQuery, variantOption: variant, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

```
}}
```

C. FIELDS

The fields parameter is used to specify the set of fields to be returned. When returning the results, only fields in the list will be included.

```
let fieldsToReturn = ["title","vPrice"]
client?.searchWithQuery(query: searchQuery, returnFields: fieldsToReturn,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

D. FACETS

Facets are the filters in the UI that allow visitors to narrow down result set based on product fields. Facet option can have 3 values:

1.**Multi-level:** The facer multilevel parameter is used to enable multi-level facets in the API response.

```
client?.searchWithQuery(query: searchQuery, facet: Facet.MultiLevel, completion:
    {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

2.**Multi-select:** This feature enables or disables the option to select multiple values within a facet or across facets for visitors.

```
client?.searchWithQuery(query: searchQuery, facet: Facet.MultiSelect, completion:
    {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

3.**Selected:** The selectedfacet parameter enables or disables the Selected Facets in the API response.

Selected facet with field id and value id:

```
let selectedIdFacet = IdFilter.init(withFieldType: "76678", fieldValue: "5001")

client?.searchWithQuery(query: searchQuery, facet:
Facet.Selected(selectedIdFacet), completion: {(response:Any?, error:Error?) ->
Void in
    //Handle response
})
```

Selected facet with field name and value name:

```
let selectedNameFacet = NameFilter.init(withFieldType: "Brand_uFilter", fieldValue:
    "Vince Camuto")

client?.searchWithQuery(query: searchQuery, facet:
Facet.Selected(selectedNameFacet), completion: {(response:Any?, error:Error?) ->
    Void in
        //Handle response
    })
```

E. FILTERING

Filtering can be performed on fields using field Id or field Name.

Three types of filters are supported:

1. Text

The text filter is used to filter products based on fields with string values such as color, gender, brand, etc. It can be defined in the API call in two ways:

- Using Field Ids

IdFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```
let idFilter = IdFilter.init(withFieldType: "76678", fieldValue: "5001")

client?.searchWithQuery(query: searchQuery, filter: idFilter, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

- Using Field Names

Again NameFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```
let nameFilter = NameFilter.init(withFieldType: "vColor_uFilter", fieldValue: "Black")

client?.searchWithQuery(query: searchQuery, filter: nameFilter, completion:
    {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

2. Range

The range filter is used to filter products based on fields with datatypes - *date*, *number* or *decimal*. It can be defined in the API in two ways:

- Using Field Id

Filter Range of type id is built using FilterIdRange class and it can be initialised with below parameters.

fieldType: The id of the field on which the text filter is applied.

lowerRange: The id of the lower limit of the range.

upperRange: The id of the upper limit of the range.

Example:

```
let idRange = FilterIdRange.init(fieldType: "76678", lowerRange: "2034",  
upperRange: "8906")  
  
client?.searchWithQuery(query: searchQuery, filter: idRange, completion:  
{(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

- Using Field Name

Filter Range of type name is built using `FilterNameRange` class and it can be initialised with below parameters.

fieldType: The name of the field on which the text filter is applied.

lowerRange: The name of the lower limit of the range.

upperRange: The name of the upper limit of the range.

Example:

```
let nameRange = FilterNameRange.init(fieldType: "vColor", lowerRange: "red",  
upperRange: "blue")  
  
client?.searchWithQuery(query: searchQuery, filter: nameRange,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

3. Multilevel

The multilevel filter is used to filter products based on categories. It can be defined in the API call in two ways:

- Using Field Ids

“CategoryIdFilter” is used to filter the results using category path comprised of category IDs

Example:

```
let categoryIdFilter = CategoryIdFilter.init()
categoryIdFilter.categories.append("FA")
categoryIdFilter.categories.append("A0485")

client?.searchWithQuery(query: searchQuery, categoryFilterPath: categoryIdFilter,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

- Using Field Names

“CategoryNameFilter” is used to filter the results using category path comprised of category Names

Example:

```
let categoryNameFilter = CategoryNameFilter.init()
categoryNameFilter.categories.append("Fashion")
categoryNameFilter.categories.append("Shirts")

client?.searchWithQuery(query: searchQuery, categoryFilterPath:
    categoryNameFilter, completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

F. MULTIPLE FILTERS

There are two types of filter operations:

- **AND**

1. Using Field IDs

MultipleIdFilter is takes 2 parameter, fieldType Id and fieldValue id.

Multiple filters can be added and 'operatorType' is set to 'AND'.

Sample:

```
let multipleIdFilter = MultipleIdFilter.init()
multipleIdFilter.operatorType = .AND
```

```

multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5001"))
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5021"))

client?.searchWithQuery(query: searchQuery, multipleFilter: multipleIdFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

2. Using Field Names

MultipleNameFilter is takes 2 parameter, fieldType name and fieldValue name.

Multiple filters can be added and 'operatorType' is set to 'AND'.

Sample:

```

let multipleNameFilter = MultipleNameFilter.init()
multipleNameFilter.operatorType = .AND
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "Black"))
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "White"))

client?.searchWithQuery(query: searchQuery, multipleFilter: multipleNameFilter,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })

```

• OR

1. Using Field IDs

MultipleIdFilter is takes 2 parameter, fieldType Id and fieldValue id.

Multiple filters can be added and 'operatorType' is set to 'OR'.

Sample:

```

let multipleIdFilter = MultipleIdFilter.init()
multipleIdFilter.operatorType = .OR

```

```

multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5001"))
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5021"))

client?.searchWithQuery(query: searchQuery, multipleFilter: multipleIdFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

2. Using Field Names

MultipleNameFilter is takes 2 parameter, fieldType name and fieldValue name.

Multiple filters can be added and 'operatorType' is set to 'OR'.

Sample:

```

let multipleNameFilter = MultipleNameFilter.init()
multipleNameFilter.operatorType = .OR
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
fieldValue: "Black"))
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
fieldValue: "White"))

client?.searchWithQuery(query: searchQuery, multipleFilter: multipleNameFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

G. SORT

The sort parameter is used to rank the products based on specified fields in the specified order.

Sort can be done on single field or multiple fields.

- Single field sort

fieldName: The field on which the sort is applied.

sortOrder: The order in which the sort is applied. This value can be "ASC" (for ascending) or "DSC" (for descending)

Sample:

```
var fieldsWithOrder = Array<FieldSortOrder>()
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "price", sortOrder: .ASC))

client?.searchWithQuery(query: searchQuery, fieldNamesWithSortOrder:
    fieldsWithOrder, completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

- Multiple field sort

Here 2 or more FieldSortOrder instances are added.

Sample:

```
var fieldsWithOrder = Array<FieldSortOrder>()
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "price", sortOrder: .ASC))
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "title", sortOrder: .DSC))

client?.searchWithQuery(query: searchQuery, fieldNamesWithSortOrder:
    fieldsWithOrder, completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

Using Auto Suggest Methods

The **Autosuggest** feature provides query suggestions which helps visitors to search faster.

Auto suggest method signature:

```
func autoSuggestWithQuery(query:AutoSuggestQuery, variant: VariantOption? = nil,
docType: DocTypeAbstract? = nil, filter: FilterAbstract? = nil, completion: @escaping (_
response:Any?,_ error:Error?) -> Void) {
    // Handle response or request
}
```

- query of type AutoSuggestQuery is mandatory and rest of the arguments are optional.

Below are few sample of invoking autoSuggestWithQuery() method with different arguments.

A. AUTOSUGGEST QUERY

AutoSuggestQuery can be initialised with 'Key' for suggestions are expected.

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

client?.autoSuggestWithQuery(query: autoSuggestQuery,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

B. VARIANT

Variants can enabled or disabled in AutoSuggest query response.

Variant Status can be set to true/false and shown below.

```

let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let variantOption = VariantOption.init(withVariantStatus: true)

client?.autoSuggestWithQuery(query: autoSuggestQuery, variant: variantOption,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })

```

C. DOCTYPE

Unbxid Autosuggest comprises of different types of suggestions that are known as doctypes. A standard Unbxid Autosuggest is segmented into five doctypes:

1. In Field

The query being typed by your visitor can belong to multiple product categories based on your product feed. The **In-fields** doctype in Autosuggest suggest groups of relevant products along with their associated field values the query may belong to.

In Field doctype with result count can be configured as shown below:

```

let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let inFieldDocType = DocTypeInField.init(resultsCount: 3)

client?.autoSuggestWithQuery(query: autoSuggestQuery, docType:
inFieldDocType, completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

If resultsCount is not set, default value 2 will be considers as results count for In Field doctype.

2. Keyword Suggestions

These are intelligent suggestions generated by Unbx'd cloud servers whose algorithm identifies the keywords from the query being typed and suggests relevant products based on your product feed accordingly.

Keyword Suggestions doctype with result count can be configured as shown below:

```
let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let keywordSuggestionsDocType =
  DocTypeKeywordSuggestions.init(resultsCount: 4)

client?.autoSuggestWithQuery(query: autoSuggestQuery, docType:
  keywordSuggestionsDocType, completion: {(response:Any?, error:Error?) -> Void
  in
    //Handle response
  })
```

If resultsCount is not set, default value 2 will be considered as results count for Keyword Suggestions doctype.

3. Top Queries

As the name suggests, this autosuggest doctype displays the frequently searched queries in your ecommerce store. These top queries are populated with the help of Unbx'd Analytics which keeps a track of your store.

Top Queries doctype with result count can be configured as shown below:

```
let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let topQueriesDocType = DocTypeTopQueries.init(resultsCount: 3)
```

```

client?.autoSuggestWithQuery(query: autoSuggestQuery, docType:
topQueriesDocType, completion: {(response:Any?, error:Error?) -> Void      in
    //Handle response
})

```

If resultsCount is not set, default value 2 will be considers as results count for Top Queries doctype.

4. Promoted Suggestions

Promoted Suggestions are documents that a customer can configure directly from merchandising console. This gives you the flexibility to manually insert keyword suggestions in autosuggest which may not be part of the default relevance results.

Promoted Suggestions doctype with result count can be configured as shown below:

```

let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let promotedSuggestionsDocType =
DocTypePromotedSuggestions.init(resultsCount: 5)

client?.autoSuggestWithQuery(query: autoSuggestQuery, docType:
promotedSuggestionsDocType, completion: {(response:Any?, error:Error?) -> Void
in
    //Handle response
})

```

If resultsCount is not set, default value 2 will be considers as results count for Promoted Suggestions doctype.

5. Popular Products

The Popular Products doctype displays popular product in your e-commerce store with thumbnail images.

Popular Products doctype with fields and result count can be configured as shown below:

```
let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let popularProductsDocType = DocTypePopularProducts.init(resultsCount: 3,
fields: ["vColor", "price"])

client?.autoSuggestWithQuery(query: autoSuggestQuery, docType:
popularProductsDocType, completion: {(response:Any?, error:Error?) -> Void    in
    //Handle response
})
```

If resultsCount is not set, default value 3 will be considers as results count for Promoted Suggestions doctype.

D. FILTERS

Filters are used in AutoSuggest method to restrict the products based on criteria passed.

Two types of filters are supported in autoSuggestWithQuery() method

1. Text

The text filter is used to filter products based on fields with string values such as color, gender, brand, etc. It can be defined in the API call in two ways:

- Using Field Ids

IdFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```
let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")
```

```
let idFilter = IdFilter.init(withFieldType: "76678", fieldValue: "5001")

client?.autoSuggestWithQuery(query: autoSuggestQuery, filter: idFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

- Using Field Names

Again NameFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```
let autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let nameFilter = NameFilter.init(withFieldType: "vColor_uFilter", fieldValue: "Black")

client?.autoSuggestWithQuery(query: autoSuggestQuery, filter: nameFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

2. Range

The range filter is used to filter products based on fields with datatypes - *date*, *number* or *decimal*. It can be defined in the API in two ways:

- Using Field Id

Filter Range of type id is built using FilterIdRange class and it can be initialised with below parameters.

fieldType: The id of the field on which the text filter is applied.

lowerRange: The id of the lower limit of the range.

upperRange: The id of the upper limit of the range.

Example:

```
llet autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let idRange = FilterIdRange.init(fieldType: "76678", lowerRange: "2034",
upperRange: "8906")

client?.autoSuggestWithQuery(query: autoSuggestQuery, filter: idRange,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

- Using Field Name

Filter Range of type name is built using FilterNameRange class and it can be initialised with below parameters.

fieldType: The name of the field on which the text filter is applied.

lowerRange: The name of the lower limit of the range.

upperRange: The name of the upper limit of the range.

Example:

```
llet autoSuggestQuery = AutoSuggestQuery.init(withKey: "Shir")

let nameRange = FilterNameRange.init(fieldType: "vColor", lowerRange: "red",
upperRange: "blue")

client?.autoSuggestWithQuery(query: autoSuggestQuery, filter: nameRange,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
}
```

Using Analytics Method

Using Analytics methods app can publish event details and user would be able track those events in the portal.

Analytics method signature:

```
func publishAnalyticsWith(analyticsDetails:AnalyticsAbstract, completion: @escaping (_
response:Any?,_ error:Error?) -> Void) {
    // Handle response or request
}
```

Below are the events and argument each event would be accepting.

1. VISITOR

Whenever a user visits the page, a visitor event is fired,

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let visitorAnalytics = VisitorAnalytics.init(withUID: uid)
client?.publishAnalyticsWith(analyticsDetails: visitorAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

Note: UID should be unique string value which is generated by app and retained through out the application life.

2. SEARCH

A search event is fired when a user types something in the search box and taps on the search button.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let searchAnalytics = SearchAnalytics.init(withUID: uid, searchKey: "Shirt")
client?.publishAnalyticsWith(analyticsDetails: searchAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

3. CATEGORY PAGE

Category Page event is fired when a user shops by department and visits a category page on the site.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let categoryQuery = CategoryNamePath.init(withCategories:
["home", "furniture", "entrywayfurniture"])
let categoryPageAnalytics = CategoryPageAnalytics.init(withUID: uid,
categoryPages: categoryQuery)

client?.publishAnalyticsWith(analyticsDetails: categoryPageAnalytics,
completion:      {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

4. PRODUCT CLICK

Whenever a user clicks on a particular product in the search or category page results, a 'click' action is generated.

Sample

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let productClickAnalytics = ProductClickAnalytics.init(withUID: uid, pid:
"2301609", searchQuery: "Socks", pageId: "Order")
client?.publishAnalyticsWith(analyticsDetails: productClickAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

5. PRODUCT ADD TO CART

Whenever a user adds a product to his cart, a 'cart' action is generated.

Sample

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let addToCartAnalytics = ProductAddToCartAnalytics.init(withUID: uid, productID:
"2301609", variantId: "231221", quantity: 2)
client?.publishAnalyticsWith(analyticsDetails: addToCartAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

6. PRODUCT ORDER

An 'order' event will be fired after an order completion, when the user comes back to the product page from the payment gateway. If a user buys multiple products in a single order, then multiple order events should be fired.

Sample

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

```

```

let orderAnalytics = ProductOrderAnalytics.init(withUID: uid, productID:
"2301609", price: 20.5, quantity: 3)
client?.publishAnalyticsWith(analyticsDetails: orderAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

7. PRODUCT DISPLAY PAGE VIEW

Product Display Page View is fired when a user visits a Product Display Page.

Sample

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let productDisplayPageAnalytics = ProductDisplayPageViewAnalytics.init(withUID:
uid, skuld: "2034")
client?.publishAnalyticsWith(analyticsDetails: productDisplayPageAnalytics,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

8. CART REMOVAL

Cart Removal event is fired when a product in cart is removed..

Sample

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let cartRemovalAnalytics = CartRemovalAnalytics.init(withUID: uid, skuld: "2034",
variantId: "231221", quantity: 2)
client?.publishAnalyticsWith(analyticsDetails: cartRemovalAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

```
}}
```

9. AUTOSUGGEST

An autosuggest event is fired when a user types something, then clicks on any of autosuggest results shown.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let autoSuggestAnalytics = AutoSuggestAnalytics.init(withUID: uid, skuld: "2034",
query: "Red Socks", docType: .INFIELD, internalQuery: "red", fieldValue: "Red
socks", fieldName: "infield1", sourceField: "color type")
client?.publishAnalyticsWith(analyticsDetails: autoSuggestAnalytics, completion:
{(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

10. RECOMMENDATION WIDGET IMPRESSION

If App is subscribed to Unbx'd Recommendations, every time the user clicks on any Recommendation widget, the API below needs to be called.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let recommendationImpressionAnalytics =
RecommendationWidgetAnalytics.init(withUID: uid, productIds:
["1692741-1758", "01692015-285", "1692908-480"])
client?.publishAnalyticsWith(analyticsDetails:
recommendationImpressionAnalytics, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
```



```
}}
```

11. SEARCH IMPRESSION

A search impression event is fired when a search results page loads for the first time, and whenever results changes on applying pagination, autoscroll, sort, and filters. For each of these action, unique Ids of the products visible on search page should be sent as payload.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let searchImpressionAnalytics = SearchImpressionAnalytics.init(withUID: uid,
query: "Shoes", productIds: ["1692741-1758", "01692015-285", "1692908-480"])
client?.publishAnalyticsWith(analyticsDetails: searchImpressionAnalytics,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

12. CATEGORY PAGE IMPRESSION

A Category Page Impression event is fired when a category page results loads for first time, and when ever the results changes on applying pagination, autoscroll, sort, and filters. For each of these action, unique Ids of the products visible on search page should be sent as payload.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")
let uid = "1409753998404-35253"

let categoryPathQuery = CategoryNamePath.init(withCategories: ["men", "shirt"])
let categoryPageImpression = CategoryPageImpressionAnalytics.init(withUID: uid,
categoryPages: categoryPathQuery, productIds:
["1692741-1758", "01692015-285", "1692908-480"])
```

```
client?.publishAnalyticsWith(analyticsDetails: categoryPageImpression,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

13. DWELLTIME

A dwellTime event is used to capture the amount of time spent on product description page.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
let uid = "1409753998404-35253"  
  
let dwellTimeAnalytics = DwellTimeAnalytics.init(withUID: uid, productID:  
"2301609", dwellTime: 60) // In Seconds  
client?.publishAnalyticsWith(analyticsDetails: dwellTimeAnalytics, completion:  
{(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

14. FACETS

A facet event is fired when a filter is applied on Search or Category pages.

Sample

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
let uid = "1409753998404-35253"  
  
let facetAnalytics = FacetAnalytics.init(withUID: uid, searchQuery: "Shirts",  
facetFields: ["Fitted"])  
client?.publishAnalyticsWith(analyticsDetails: facetAnalytics, completion:  
{(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

Using Browse Methods (Category pages)

Browse methods operates on Category fields query which is configured part of Browse Query.

Search method signature:

```
func browseWithQuery(query:BrowseQuery, variantOption: VariantOption? = nil,
returnFields:Array<String>? = nil, facet: Facet = .None, filter: FilterAbstract? = nil,
categoryFilterPath: CategoryFilterAbstract? = nil, multipleFilter: MultipleFilterAbstract?
= nil, fieldNamesWithSortOrder: Array<FieldSortOrder>? = nil, completion: @escaping
(_ response:Any?,_ error:Error?) -> Void) {
    // Handle response or request
}
```

* 'query' of type 'BrowseQuery' is mandatory argument and rest are optional which are used as needed.

Lets see how these arguments can be composed and passed in browseWithQuery() method invocation.

A. BROWSEQUERY

BrowseQuery consists of Category path or field details parameter and few other optional parameters.

Browse Query using Category Query can be build as shown below:

- Using Field Ids

```
let categoryQuery = CategoryIdFields.init(fieldType: "76678", fieldValue: "5001")
```

```
let browseQuery = BrowseQuery.init(withCategoryQuery: categoryQuery)
```

```
client?.browseWithQuery(query: browseQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})
```

- Using Field Names

```
let categoryQuery = CategoryNameFields.init(fieldType: "vColor", fieldValue:
"Black")

let browseQuery = BrowseQuery.init(withCategoryQuery: categoryQuery)

client?.browseWithQuery(query: browseQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})
```

- Using Page Ids

```
let categoryQuery = CategoryIdPath.init(withCategories: ["FA", "FA0484"])

let browseQuery = BrowseQuery.init(withCategoryQuery: categoryQuery)

client?.browseWithQuery(query: browseQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})
```

- Using Page Names

```
let categoryQuery = CategoryIdPath.init(withCategories:
["Fashion", "Shoes", "Sneakers", "Athletic shoes"])

let browseQuery = BrowseQuery.init(withCategoryQuery: categoryQuery)
```

```

client?.browseWithQuery(query: browseQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})

```

- Optional preference in Browse Category

Below are few parameters which are part of Browse Query which can applied for fewer response.

```

let categoryQuery = CategoryIdPath.init(withCategories:
["Fashion","Shoes","Sneakers","Athletic shoes"])

let browseQuery = BrowseQuery.init(withCategoryQuery: categoryQuery)

//Optional parameters
browseQuery.rowsCount = 20
browseQuery.pageIndex =
browseQuery.responseFormat = ResponseFormat.JSON
browseQuery.spellCheck = true
browseQuery.analytics = true
browseQuery.showStatsForField = "price"

client?.browseWithQuery(query: browseQuery, completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})

```

B. VARIANTS

Variants parameter enables or disables variants in the API response. It can take two values: “true” or “false”. Default value is “false”.

```

        let variant = VariantOption.init(withVariantStatus: true)
        client?.browseWithQuery(query: browseQuery, variantOption: variant,
completion:      {(response:Any?, error:Error?) -> Void in
                        //Handle response
                    })

```

If you want to get more than one variants in the API response, you can use 'variantCount' parameter. It can have any numerical value (eg, 1,2,3,etc) or ".max" (to get all the variants).

With Variant count value

```

        let variant = VariantOption.init(withVariantStatus: true, variantCount: 2)
        client?.browseWithQuery(query: browseQuery, variantOption: variant,
completion:      {(response:Any?, error:Error?) -> Void in
                        //Handle response
                    })

```

With all Variants

```

        let variant = VariantOption.init(withVariantStatus: true, variantCount: .max)
        client?.browseWithQuery(query: browseQuery, variantOption: variant,
completion:      {(response:Any?, error:Error?) -> Void in
                        //Handle response
                    })

```

C. FIELDS

The fields parameter is used to specify the set of fields to be returned. When returning the results, only fields in the list will be included.

```

        let fieldsToRerturn = ["title", "vPrice"]
        client?.browseWithQuery(query: browseQuery, returnFields: fieldsToRerturn,
completion: {(response:Any?, error:Error?) -> Void in
                    //Handle response
                })

```

D. FACETS

Facets are the filters in the UI that allow visitors to narrow down result set based on product fields. Facet option can have 3 values:

1.**Multi-level**: The facer multilevel parameter is used to enable multi-level facets in the API response.

```
client?.browseWithQuery(query: browseQuery, facet: Facet.MultiLevel,
completion:      {(response:Any?, error:Error?) -> Void in
                //Handle response
            })
```

2.**Multi-select**: This feature enables or disables the option to select multiple values within a facet or across facets for visitors.

```
client?.browseWithQuery(query: browseQuery, facet: Facet.MultiSelect,
completion:      {(response:Any?, error:Error?) -> Void in
                //Handle response
            })
```

3.**Selected**: The selectedfacet parameter enables or disables the Selected Facets in the API response.

Selected facet with field id and value id:

```
let selectedIdFacet = IdFilter.init(withFieldType: "76678", fieldValue: "5001")

client?.browseWithQuery(query: browseQuery, facet:
Facet.Selected(selectedIdFacet), completion: {(response:Any?, error:Error?) ->
Void in
        //Handle response
    })
```

Selected facet with field name and value name:

```

let selectedNameFacet = NameFilter.init(withFieldType: "Brand_uFilter", fieldValue:
    "Vince Camuto")

client?.browseWithQuery(query: browseQuery, facet:
    Facet.Selected(selectedNameFacet), completion: {(response:Any?,
error:Error?) -> Void in
    //Handle response
})

```

E. FILTERING

Filtering can be performed on fields using field Id or field Name.

Three types of filters are supported:

1. Text

The text filter is used to filter products based on fields with string values such as color, gender, brand, etc. It can be defined in the API call in two ways:

- Using Field Ids

IdFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```

let idFilter = IdFilter.init(withFieldType: "76678", fieldValue: "5001")

client?.browseWithQuery(query: browseQuery, filter: idFilter, completion:
    {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

- Using Field Names

Again NameFilter can be formed with 2 parameters.

fieldType: The id of the field on which the text filter is applied.

fieldValue: The id of the value on which the results are filtered.

Example:

```
let nameFilter = NameFilter.init(withFieldType: "vColor_uFilter", fieldValue: "Black")

client?.browseWithQuery(query: browseQuery, filter: nameFilter, completion:
    {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

2. Range

The range filter is used to filter products based on fields with datatypes - *date*, *number* or *decimal*. It can be defined in the API in two ways:

- Using Field Id

Filter Range of type id is built using `FilterIdRange` class and it can be initialised with below parameters.

fieldType: The id of the field on which the text filter is applied.

lowerRange: The id of the lower limit of the range.

upperRange: The id of the upper limit of the range.

Example:

```
let idRange = FilterIdRange.init(fieldType: "76678", lowerRange: "2034",
    upperRange: "8906")

client?.browseWithQuery(query: browseQuery, filter: idRange, completion:
    {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

- Using Field Name

Filter Range of type name is built using `FilterNameRange` class and it can be initialised with below parameters.

fieldType: The name of the field on which the text filter is applied.

lowerRange: The name of the lower limit of the range.

upperRange: The name of the upper limit of the range.

Example:

```
let nameRange = FilterNameRange.init(fieldType: "vColor", lowerRange: "red",  
    upperRange: "blue")  
  
client?.browseWithQuery(query: browseQuery, filter: nameRange,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

3. Multilevel

The multilevel filter is used to filter products based on categories. It can be defined in the API call in two ways:

- Using Field Ids

“CategoryIdFilter” is used to filter the results using category path comprised of category IDs

Example:

```
let categoryIdFilter = CategoryIdFilter.init()  
categoryIdFilter.categories.append("FA")  
categoryIdFilter.categories.append("A0485")  
  
client?.browseWithQuery(query: browseQuery, categoryFilterPath:  
categoryIdFilter, completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

- Using Field Names

“CategoryNameFilter” is used to filter the results using category path comprised of category Names

Example:

```
let categoryNameFilter = CategoryNameFilter.init()
categoryNameFilter.categories.append("Fashion")
categoryNameFilter.categories.append("Shirts")

client?.browseWithQuery(query: browseQuery, categoryFilterPath:
    categoryNameFilter, completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
    })
```

F. MULTIPLE FILTERS

There are two types of filter operations:

- **AND**

1. Using Field IDs

MultipleIdFilter takes 2 parameters, fieldType Id and fieldValue id.

Multiple filters can be added and 'operatorType' is set to 'AND'.

Sample:

```
let multipleIdFilter = MultipleIdFilter.init()
multipleIdFilter.operatorType = .AND
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5001"))
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
"5021"))

client?.browseWithQuery(query: browseQuery, multipleFilter: multipleIdFilter,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
    })
```

2. Using Field Names

MultipleNameFilter is takes 2 parameter, fieldType name and fieldValue name.
Multiple filters can be added and 'operatorType' is set to 'AND'.

Sample:

```
let multipleNameFilter = MultipleNameFilter.init()
multipleNameFilter.operatorType = .AND
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "Black"))
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "White"))

client?.browseWithQuery(query: browseQuery, multipleFilter: multipleNameFilter,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

- OR

1. Using Field IDs

MultipleIdFilter is takes 2 parameter, fieldType Id and fieldValue id.
Multiple filters can be added and 'operatorType' is set to 'OR'.

Sample:

```
let multipleIdFilter = MultipleIdFilter.init()
multipleIdFilter.operatorType = .OR
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
    "5001"))
multipleIdFilter.filters.append(IdFilter.init(withFieldType: "76678", fieldValue:
    "5021"))

client?.browseWithQuery(query: browseQuery, multipleFilter: multipleIdFilter,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

2. Using Field Names

MultipleNameFilter is takes 2 parameter, fieldType name and fieldValue name.

Multiple filters can be added and 'operatorType' is set to 'OR'.

Sample:

```
let multipleNameFilter = MultipleNameFilter.init()
multipleNameFilter.operatorType = .OR
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "Black"))
multipleNameFilter.filters.append(NameFilter.init(withFieldType: "vColor_uFilter",
    fieldValue: "White"))

client?.browseWithQuery(query: browseQuery, multipleFilter: multipleNameFilter,
    completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

G. SORT

The sort parameter is used to rank the products based on specified fields in the specified order.

Sort can be done on single field or multiple fields.

- Single field sort

fieldName: The field on which the sort is applied.

sortOrder: The order in which the sort is applied. This value can be "ASC" (for ascending) or "DSC" (for descending)

Sample:

```
var fieldsWithOrder = Array<FieldSortOrder>()
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "price", sortOrder: .ASC))

client?.browseWithQuery(query: browseQuery, fieldNamesWithSortOrder:
    fieldsWithOrder, completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

- Multiple field sort

Here 2 or more FieldSortOrder instances are added.

Sample:

```
var fieldsWithOrder = Array<FieldSortOrder>()
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "price", sortOrder: .ASC))
fieldsWithOrder.append(FieldSortOrder.init(fieldName: "title", sortOrder: .DSC))

client?.browseWithQuery(query: browseQuery, fieldNamesWithSortOrder:
    fieldsWithOrder, completion: {(response:Any?, error:Error?) -> Void in
        //Handle response
    })
```

Using Recommendations Widget

Unboxd Recommendations offers a wide-range of widgets for every page. Recommendations API returns product details for options like MoreLikeThis, RecentlyViewed etc.

Recommendations method signature:

```
func fetchRecommendationFor(recommendationQuery:RecommendationsAbstract,
completion: @escaping (_ response:Any?,_ error:Error?) -> Void) {
    // Handle response or request
}
```

Below are the different recommendation types and its arguments.

1. RECOMMENDED FOR YOU

The Recommended For You method returns recommendations based on the visitor's interaction history on the online store or app.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let forYouQuery = RecommendedForYourRecommendations.init(WithUID: uid,
region: "USA", currencyFormat: "USD", responseFormat: .JSON)

client?.fetchRecommendationFor(recommendationQuery: forYouQuery,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

Note: UID should be unique string value which is generated by app and retained through out the application life.

- region : ISO standard country code.
- currencyFormat: ISO standard currency code.
- responseFormat: .JSON or XML

2. RECENTLY VIEWED

The Recently Viewed method recommends products that were recently viewed by a visitor.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let recentlyViewedQuery = RecentlyViewedRecomendations.init(WithUID: uid,
productID: "2312314", region: "USA")
client?.fetchRecommendationFor(recommendationQuery: recentlyViewedQuery,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

3. MORE LIKE THESE

The "More Like These" method is built to recommend products similar to the one being viewed on the PDP (Product Detail Page).

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let moreLikeThisQuery = MoreLikeThisRecomendations.init(WithUID: uid,
productID: "2312314", region: "USA")
```



```
client?.fetchRecommendationFor(recommendationQuery: moreLikeThisQuery,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

4. VIEWED ALSO VIEWED

As the name suggests, this method recommends products viewed by other visitors.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
  
let uid = "1409753998404-35253"  
  
let alsoViewedQuery = ViewedAlsoViewedRecomendations.init(WithUID: uid,  
productID: "2312314", region: "USA")  
client?.fetchRecommendationFor(recommendationQuery: alsoViewedQuery,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

5. BOUGHT ALSO BOUGHT

imilar to the "Viewed also Viewed" method, the Bought also Bought method recommends products bought by other visitors.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
  
let uid = "1409753998404-35253"  
  
let alsoBoughtdQuery = BoughtAlsoBoughtRecomendations.init(WithUID: uid,  
productID: "2312314", region: "USA")
```

```
client?.fetchRecommendationFor(recommendationQuery: alsoBoughtdQuery,  
completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

6. CART RECOMMENDATIONS

This method recommends complementary products on the "Cart page" for those present in the visitor's cart.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
  
let uid = "1409753998404-35253"  
  
let cartRecommendationQuery = CartRecomendations.init(WithUID: uid, region:  
"USA")  
client?.fetchRecommendationFor(recommendationQuery:  
cartRecommendationQuery, completion: {(response:Any?, error:Error?) -> Void in  
    //Handle response  
})
```

7. HOMEPAGE TOP SELLERS

This method recommends top selling products bought from the homepage.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")  
  
let uid = "1409753998404-35253"  
  
let homePageTopSellerQuery =  
HomePageTopSellersRecomendations.init(WithUID: uid, region: "USA")
```

```

client?.fetchRecommendationFor(recommendationQuery:
homePageTopSellerQuery, completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})

```

8. CATEGORY TOP SELLERS

This method recommends top selling products from a specific category.

Sample:

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let categoryTopSellerQuery = CategoryTopSellersRecomendations.init(WithUID:
uid, region: "USA", categoryName: "<CategoryName>")
client?.fetchRecommendationFor(recommendationQuery:
categoryTopSellerQuery, completion: {(response:Any?, error:Error?) ->
Void in
    //Handle response
})

```

9. PDP TOP SELLERS

This method recommends top selling products from a specific product.

Sample:

```

let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let pdpTopSellerQuery = PDPTopSellersRecomendations.init(WithUID: uid, region:
"USA")
client?.fetchRecommendationFor(recommendationQuery: pdpTopSellerQuery,
completion: {(response:Any?, error:Error?) -> Void in

```

```
        //Handle response
    })
```

10. BRAND TOP SELLERS

This method recommends top selling products from a specific brand.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let brandTopSellerQuery = BrandTopSellersRecomendations.init(WithUID: uid,
region: "USA", brandName: "Nike")
client?.fetchRecommendationFor(recommendationQuery: brandTopSellerQuery,
completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```

11. COMPLETE THE LOOK

The Complete the Look method showcases curated products on a Product Display Page (PDP) that are usually associated with the original product.

Sample:

```
let client = Client.init(siteKey: "SITE_KEY", apiKey: "API_KEY")

let uid = "1409753998404-35253"

let completeTheLookQuery = CompleteTheLookRecomendations.init(WithUID: uid,
productID: "2312314", region: "USA")
client?.fetchRecommendationFor(recommendationQuery:
completeTheLookQuery, completion: {(response:Any?, error:Error?) -> Void in
    //Handle response
})
```