iOS SDK

You can send data from your iOS and tvOS apps to Treasure Data, using our iOS SDK library. With this SDK, you can import the events on your applications into Treasure Data. Technically, this library supports iOS 7 and later, but we only execute OS coverage tests for iOS 8, 9, 10, 11, 12, 13, and 14. This SDK also supports Apple tvOS 12 and up.

There is an alternative SDK written in Swift https://github.com/recruit-lifestyle/TreasureDataSDK. However, it does not support the current GDPR functionality in the mainstream TD SDKs.

Prerequisites and Requirements

- Basic knowledge of iOS Development (Xcode, CocoaPods)
- Basic knowledge of Treasure Data
- iOS 7 or later
- Ensure compliance with GDPR
- Access to Treasure Data iOS SDK releases

Treasure Data recommends that you implement any new features or functionality at your site using the Treasure Data JavaScript SDK version 3 Beta. It manages cookies differently. Be aware when referring to most of these articles that you need to define the suggested event collectors and Treasure Data JavaScript SDK version 3 calls in your solutions.

For example, change //cdn.treasuredata.com/sdk/2.5/td.min.js to //cdn.treasuredata.com/sdk/3.0.0-beta/td.min.js.

Install iOS SDK

Install the Treasure Data iOS SDK Library

Treasure Data recommends that you use CocoaPods to install Treasure Data iOS SDK.

CocoaPods

1. Install CocoaPods to your computer.

```
$ gem install cocoapods
```

2. Add the following line to your Podfile, where <version> is 0.8.1 or later:

```
pod 'TreasureData-iOS-SDK', '= <version>'
```

3. Optionally, if you use the SDK in Swift, add the following line to your Podfile:

```
use_frameworks!
```

4. Run the pod install:

```
$ pod install
```

Remember to reopen your project by opening .xcworkspace file instead of .xcodeproj file

Framework

1. Locate Treasure Data iOS SDK.
2. Download the TreasureData-iOS-SDK.framework.
3. Move the TreasureData-iOS-SDK.framework.zip and the ~~libz~~ library into your project.

Initialize the Treasure Data iOS SDK Library

- Treasure Data's Guide (most parts are overlapped with this README)
- API Reference

Import SDK header file

```
#import <TreasureData-iOS-SDK/TreasureData.h>
```

Register your TreasureData API key

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    [TreasureData initializeWithApiKey:@"your_api_key"];
}
```

We recommend using a write-only API key for the SDK.

1. Login into the Treasure Data Console.
2. Visit your Profile page.
3. Insert your password under the 'API Keys' panel.
4. Under 'Write-Only API keys', copy the API key or click on 'Generate New' and copy the new API key.

Optionally Enable Tracking of Personal Information

To comply with data privacy regulations in various domains, and specifically the EU's GDPR, the Treasure Data iOS SDK does not collect certain event metadata that is personally identifiable. Specifically, the following information is not collected by default:

- `td_uuid` - client's identifier, unique to this installation on this device

The `td_uuid` is needed if you want to track individual users and analyze their data within and across user sessions, associate the tracked behavior with a real-world individual. Learn more about UUID for iOS SDK.

1. Review your data collection policy with your company's data privacy officer and legal counsel to determine what if any personal information you should collect.

2. If you decide to enable tracking of individuals, we also recommend that you integrate with a consent management system to track individual user opt-ins to tracking.
3. After you have determined the user consent, you can enable the collection of personal data. For example:

```
[[TreasureData sharedInstance] enableAutoAppendUniqId];
```

4. Test your setup and configuration.
5. Review the information you are collecting to ensure that you have the personal information you intended and nothing more.

Add an Event to a Local Buffer

To add an event to a local buffer, you can call Treasure Data's ~~addEvent~~ or ~~addEventWithCallback~~ API.

```
- (IBAction)clickButton:(id)sender {
    [[TreasureData sharedInstance] addEventWithCallback:@{
                        @"name": @"boo bar",
                                          @"age": @42,
                        @"comment": @"hello meg"
                    }
                    database:@"testdb"
                       table:@"demotbl"
                   onSuccess:^(){
                       NSLog(@"addEvent: success");
                   }
                    onError:^(NSString* errorCode, NSString* message) {
                        NSLog(@"addEvent: error. errorCode=%@, message=%@", errorCode, message);
                    }];

   // Or, simply...
   // [[TreasureData sharedInstance] addEvent:@{
   // @"name": @"boo bar",
   // @"age": @42,
   // @"comment": @"hello meg"
   // }
   // database:@"testdb"
   // table:@"demotbl"];
```

Specify the database and table to which you want to import the events. The total length of the database and table must be less than 129 characters.

Upload Buffered Events to TreasureData

To upload events buffered events to Treasure Data, you can call Treasure Data's ~~uploadEvents~~ or ~~uploadEventsWithCallback~~ API.

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
       __block UIBackgroundTaskIdentifier bgTask = [application beginBackgroundTaskWithExpirationHandler:^{
              [application endBackgroundTask:bgTask];
              bgTask = UIBackgroundTaskInvalid;
       }];

     // You can call this API to uplaod buffered events whenever you want.
         [[Treasure Data sharedInstance] uploadEventsWithCallback:^() {
                        [application endBackgroundTask:bgTask];
                        bgTask = UIBackgroundTaskInvalid;
                }
                onError:^(NSString *code, NSString *msg) {
                        [application endBackgroundTask:bgTask];
                        bgTask = UIBackgroundTaskInvalid;
                }
         ];

     // Or, simply...
     // [[Treasure Data sharedInstance] uploadEvents];
```

It depends on the characteristic of your application when to upload and how often to upload buffered events. We recommend the following upload opportunities:

- When the current screen is closing or moving to the background
- When closing the application

The sent events are buffered for a few minutes before they get imported into Treasure Data storage.

In tvOS, cache storage is stored in the cache directory which can be purged at any time. It is highly recommended that you call upload events APIs as frequently as possible to prevent loss of data.

Retry Uploading and Deduplication

The SDK imports events in one style with the combination of these features:

- This SDK keeps buffered events by adding unique keys and retries to upload them until confirming the events are uploaded and stored on the server-side (at least once)
- The server side remembers the unique keys of all events within the past 1 hour by default and can prevent duplicate imports.

Deduplication is a best effort system that identifies a duplicate record if a record with the same identifier is seen in the same dataset, within the last hour at most or within the last 4096 records, whichever comes first.

Also, you can append session information to each event using Treasure Data's class methods, Start and End Session.

Default Values

Set a default value if you want to add an event to a table, a database, or for any table or database to automatically set a value for a key. If you have multiple default values set to the same key, the newly added event will have the default value applied and override in the following order:

1. The default value targeting all tables and databases is applied first.
2. The default value targeting all tables in a database is then be applied.
3. The default value targeting the table to which the event is added is then applied.
4. The default value targeting the table and database to which the event is added is then applied.
5. If the event has a value for the key, that value will override all default values.

Set a Default Value

```
[[TreasureData sharedInstance] setDefaultValue:@"Value" forKey:@"key" database:nil table:nil]; // Targeting all
databases and tables
[[TreasureData sharedInstance] setDefaultValue:@"Value" forKey:@"key" database:"database_name" table:nil]; //
Targeting all tables of database "database_name"
[[TreasureData sharedInstance] setDefaultValue:@"Value" forKey:@"key" database:nil table:"table_name"]; //
Targeting all tables with "table_name"
[[TreasureData sharedInstance] setDefaultValue:@"Value" forKey:@"key" database:"database_name" table:"
table_name"]; // Targeting table "table_name" of database "database_name"
```

Get a Default Value

```
NSString *defaultValue = [[TreasureData sharedInstance] defaultValueForKey:@"key" database:"database_name"
table:"table_name"]; // Get default value for key targeting database "database_name" and table "table_name".
```

Remove a Default Value

```
[[TreasureData sharedInstance] removeDefaultValueForKey:@"key" database:"database_name" table:"table_name"]; //
Only remove default values targeting database "database_name" and table "table_name".
```

Start and End Session

When you call ~~startSession~~ method, the SDK generates a session ID that's kept until ~~endSession~~ is called. The session id is output as a column name "td_session_id". Also, ~~startSession~~ and ~~endSession~~ methods add an event that includes ~~{"td_session_event":"start" or "end"}~~ .

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
                [TreasureData initializeWithApiKey:@"your_api_key"];
                [[TreasureData sharedInstance] setDefaultDatabase:@"testdb"];
                [[TreasureData sharedInstance] startSession:@"demotbl"];
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
                [[TreasureData sharedInstance] endSession:@"demotbl"];

                __block UIBackgroundTaskIdentifier bgTask = [application
beginBackgroundTaskWithExpirationHandler:^{
                                [application endBackgroundTask:bgTask];
                                bgTask = UIBackgroundTaskInvalid;
                }];

                [[TreasureData sharedInstance] uploadEventsWithCallback:^() {
                                                [application endBackgroundTask:bgTask];
                                                bgTask = UIBackgroundTaskInvalid;
                }
                onError:^(NSString *code, NSString *msg) {
                                [application endBackgroundTask:bgTask];
                                bgTask = UIBackgroundTaskInvalid;
                }
                // Outputs =>>
                // [{"td_session_id":"cad88260-67b4-0242-1329-2650772a66b1",
                // "td_session_event":"start", "time":1418880000},
                //
                // {"td_session_id":"cad88260-67b4-0242-1329-2650772a66b1",
                // "td_session_event":"end", "time":1418880123}
                // ]
];
```

If you want to handle the following case, use a pair of class methods ~~startSession~~ and ~~endSession~~ for global session tracking

- The user opens the application and starts session tracking using ~~startSession~~. Let's call this session#0
- The user moves to the home screen and finishes the session using ~~endSession~~
- The user reopens the application and restarts session tracking within the default 10 seconds. But you want to deal with this new session as the same session as session#0

```
- (void)applicationDidBecomeActive:(UIApplication *)application
{
                [TreasureData startSession];
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
                [TreasureData endSession];
}
```

In this case, you can get the current session ID using the ~~getSessionId~~ class method.

```
- (void)applicationDidBecomeActive:(UIApplication *)application
{
                [TreasureData startSession];
        NSLog(@"Session ID=%@", [TreasureData getSessionId]);
}
```

Track the Application First Running

Detect if it's the first running with the ~~isFirstRun~~ method and then clear the flag with ~~clearFirstRun~~.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
:
if ([[TreasureData sharedInstance] isFirstRun]) {
[[TreasureData sharedInstance] addEventWithCallback:@{ @"event": @"installed" }
database:@"testdb"
table:@"demomegtbl"
onSuccess:^(){
[[TreasureData sharedInstance] uploadEventsWithCallback:^() {
[[TreasureData sharedInstance] clearFirstRun];
}
onError:^(NSString* errorCode, NSString* message) {
NSLog(@"uploadEvents: error. errorCode=%@, message=%@", errorCode, message);
}
];
}
onError:^(NSString* errorCode, NSString* message) {
NSLog(@"addEvent: error. errorCode=%@, message=%@", errorCode, message);
}];
}
```

Understanding Error Codes

~~addEventWithCallback~~ and ~~uploadEventsWithCallback~~ methods call back ~~onError~~ block with ~~errorCode~~ argument. This argument is useful to know the cause type of error. There are the following error codes:

- ~~init_error~~: The initialization failed.
- ~~invalid_param~~: The parameter passed to the API was invalid
- ~~invalid_event~~: The event was invalid
- ~~data_conversion~~: Failed to convert the data to/from JSON
- ~~storage_error~~: Failed to read/write data in the storage
- ~~network_error~~: Failed to communicate with the server due to network problem
- ~~server_response~~: The server returned an error response

Additional Configurations

Endpoint

The API endpoint (default: https://in.treasuredata.com) can be modified using ~~initializeApiEndpoint~~ class method.

```
[TreasureData initializeApiEndpoint:@"https://specifying-another-endpoint.com"];
[TreasureData initializeWithApiKey:@"your_api_key"];
```

Encryption Key

If you've set an encryption key using ~~initializeEncryptionKey~~ class method, our SDK saves the events data as encrypted when called ~~addEvent~~ or ~~addEvent WithCallback~~ methods.

```
[TreasureData initializeEncryptionKey:@"hello world"];

[[TreasureData sharedInstance] addEventWithCallback: ....];
```

Default Database

```
[[TreasureData sharedInstance] setDefaultDatabase:@"testdb"];

[[TreasureData sharedInstance] addEventWithCallback:@{ @"event": @"clicked" } table:@"demotbl"]
```

Automatically Add Device UUID to Each Event

To add the UUID of the device to each event automatically, call ~~enableAutoAppendUniqId~~ . This value won't change until the application is uninstalled or ~~resetUniqId~~ is called.

```
[[TreasureData sharedInstance] enableAutoAppendUniqId];
```

This call outputs the value as the column name ~~td_uuid~~.

Get and Reset UUID

You can get the current UUID (~~td_uuid~~) using the following API. Remember that this UUID will change if ~~resetUniqId~~ is called.

```
NSString *td_uuid = [[TreasureData sharedInstance] getUUID];
```

You can also reset the UUID (~~td_uuid~~) using the following API.

```
[[TreasureData sharedInstance] resetUniqId];
```

Automatically Add UUID to an Event Record

Call ~~enableAutoAppendRecordUUID~~ to automatically add a UUID for an event record. Each event has a different UUID.

```
[[TreasureData sharedInstance] enableAutoAppendRecordUUID];

// If you want to customize the column name, pass it to the API
[[TreasureData sharedInstance] enableAutoAppendRecordUUID:@"my_record_uuid"];
```

It outputs the value as a column name ~~record_uuid~~ by default.

Automatically Add Advertising ID to an Event Record

Call ~~enableAutoAppendAdvertisingIdentifier~~ to automatically add an Advertising Id to each event record.

You must link the Ad Support framework in Link Binary With Libraries build phase for this feature to work. Users must also not turn on the Limit Ad Tracking feature in their iOS device, otherwise, Treasure Data sends a zero-filled string as the advertising id (the value we get from the Ad Support framework).

Starting in iOS 14, you must explicitly request the user's permission for advertising identifier using the AppTrackingTransparency framework. Consult Apple's official documentation for AppTrackingTransparency on how to implement this requirement.

If you turn on this feature, keep in mind that you will have to declare the correct reason for getting an advertising identifier when you submit your app for review to the App Store.

```
[[TreasureData sharedInstance] enableAutoAppendAdvertisingIdentifier];

// If you want to customize the column name, pass it to the API
[[TreasureData sharedInstance] enableAutoAppendAdvertisingIdentifier:@"custom_ad_id_column"];
```

It outputs the value as a column name ~~td_maid~~ by default.

Automatically Add Device Model Information to Each Event

Call ~~enableAutoAppendModelInformation~~ to automatically add device model information to each event.

```
[[TreasureData sharedInstance] enableAutoAppendModelInformation];
```

It outputs the following column names and values:

- ~~td_device:~~ UIDevice.model
- ~~td_model:~~ UIDevice.model
- ~~td_os_ver:~~ UIDevice.model.systemVersion
- ~~td_os_type:~~ "iOS"

Automatically Add Application Version Information to Each Event

Call ~~enableAutoAppendAppInformation~~ to automatically add application version information to each event.

```
[[TreasureData sharedInstance] enableAutoAppendAppInformation];
```

It outputs the following column names and values:

- ~~td_app_ver:~~ Core Foundation key ~~CFBundleShortVersionString~~
- ~~td_app_ver_num:~~ Core Foundation key ~~CFBundleVersion~~

Automatically Add Locale Configuration Information to Each Event

Call ~~enableAutoAppendLocaleInformation~~ to automatically add locale configuration information to each event.

```
[[TreasureData sharedInstance] enableAutoAppendLocaleInformation];
```

It outputs the following column names and values:

- ~~td_locale_country:~~ ~~[[NSLocale currentLocale] objectForKey: NSLocaleCountryCode]~~
- ~~td_locale_lang:~~ ~~[[NSLocale currentLocale] objectForKey: NSLocaleLanguageCode]~~

Use Server-Side Upload Timestamp

Call ~~enableServerSideUploadTimestamp~~ to use a server-side upload timestamp that is recorded when your application calls ~~addEvent.~~

```
// Use server side upload time as `time` column
[[TreasureData sharedInstance] enableServerSideUploadTimestamp];

// Add server side upload time as a customized column name
[[TreasureData sharedInstance] enableServerSideUploadTimestamp:@"server_upload_time"];
```

Enable/Disable Debug Log

```
[TreasureData enableLogging];
```

```
[TreasureData disableLogging];
```

Automatic Event Tracking

All of these are **disabled by default*, you have to explicitly enable it for each category.

Application Lifecycle Events

The following code can be used to enable lifecycle events

```
[[TreasureData sharedInstance] enableAppLifecycleEvent];
```

There are three app lifecycle event types that you can track: ~~TD_IOS_APP_OPEN~~, ~~TD_IOS_APP_INSTALL,~~ and ~~TD_IOS_APP_UPDATE.~~ These event types output to the ~~td_ios_event~~ column. The following is an example of a tracked install event.

```
"td_ios_event" = "TD_IOS_APP_INSTALL";
"td_app_ver" = "1.1";
"td_app_ver_num" = 2;
```

In-App Purchase Events

TreasureData SDK is able to automatically track IAP ~~SKPaymentTransactionStatePurchased~~ event without having to write your own transaction observer.

```
[[TreasureData sharedInstance] enableInAppPurchaseEvent];
```

There is a subtle difference between InAppPurchaseEvent, AppLifecycleEvent, and customEvent. The other two are persistent settings; their statuses are saved across app launches. `inAppPurchaseEvent` behaves like an ordinary object option and is not saved. You have to enable it after initialize your new `TreasureData` instance (only `sharedInstance` with `initializeWithApiKey()`). The following is an example of an IAP event.

```
"td_ios_event": "TD_IOS_IN_APP_PURCHASE",
"td_iap_transaction_identifier": "1000000514091400",
"td_iap_transaction_date": "2019-03-28T08:44:12+07:00",
"td_iap_quantity": 1,
"td_iap_product_identifier": "com.yourcompany.yourapp.yourproduct", ,
"td_iap_product_price": 0.99,
"td_iap_product_localized_title": "Your Product Title",
"td_iap_product_localized_description": "Your Product Description",
"td_iap_product_currency_code": "USD", // this is only available on iOS 10 and above
```

Treasure Data will do a separated ~~SKProductsRequest~~ to get full product's information. If the request fails, fields with "td_iap_product_" prefix will be null. The ~~currency_code~~ is only available from iOS 10 onwards.

Profile API

fetchUserSegments

This feature is not enabled on accounts by default; contact support for more information.

You must set cdpEndpoint property of TreasureData's sharedInstance.

Usage example:

```
// Set cdpEndpoint when initialize TreasureData
[[TreasureData sharedInstance] setCdpEnpoint: @"[your cdp endpoint goes here]"]

// Call fetchUserSegments to get user segments as NSArray

NSArray *audienceTokens = @[@"Your Profile API (Audience) Token here"];
NSDictionary *keys = @{@"your_key": @"your_value"};
NSDictionary<TDRequestOptionsKey, id> *options = @{
        TDRequestOptionsTimeoutIntervalKey: [NSNumber numberWithInteger: 10],
        TDRequestOptionsCachePolicyKey: [NSNumber numberWithUnsignedInteger:
NSURLRequestReloadIgnoringCacheData]
};
[[TreasureData sharedInstance] fetchUserSegments:audienceTokens
                                                                        keys:keys
                                                                  options:options
                                                        completionHandler:^(NSArray * _Nullable
jsonResponse, NSError * _Nullable error) {
        NSLog(@"fetchUserSegments jsonResponse: %@", jsonResponse);
        NSLog(@"fetchUserSegments error: %@", error);
}];
```

GDPR Compliance

The SDK provides some convenient methods to easily opt-out of tracking the device entirely without having to resort to many cluttered if-else statements:

```
// Opt-out of your own events
[[TreasureData sharedInstance] disableCustomEvent];
// Opt-out of TD generated events
[[TreasureData sharedInstance] disableAppLifecycleEvent];
[[TreasureData sharedInstance] disableInAppPurchaseEvent];
```

These can be opted back in by calling ~~enableCustomEvent~~ or ~~enableAppLifecycleEvent~~. Note that these settings are saved persistently, so it survives across app launches. Generally these methods should be called when reflecting your user's choice, not on every time initializing the SDK. By default custom events are enabled and app lifecycles events are disabled.

- Use ~~resetUniqId~~ to reset the identification of device on subsequent events. ~~td_uuid~~ will be randomized to another value and an extra event is captured with ~~{"td_ios_event": "forget_device_id", "td_uuid": <old_uuid>}~~ to the ~~defaultTable~~.

tvOS

This SDK supports Apple tvOS version 12 and up. APIs and their behaviors are largely the same as being used in an iOS application, except:

> In tvOS, cache storage is stored in the cache directory which can be purged at any time. It is highly recommended to call upload events APIs as frequently as possible to prevent loss of data.

Troubleshooting

With "Data Protection" enabled, TD iOS SDK occasionally crashes

If your app calls the SDK's API such as ~~TreasureData#endSession~~ in ~~UIApplicationDelegate applicationDidEnterBackground~~, check to see if the app calls the SDK's API several seconds after iOS is locked. If yes, make other tasks that take time and are called prior to the SDK's API run in the background.

```
- (void)applicationWillResignActive:(UIApplication *)application
{
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
// Some tasks that can take more than 10 seconds.
});
}
```

Usage in Swift

See this example project (https://github.com/treasure-data/td-ios-sdk/tree/master/TreasureDataExampleSwift) for details.

Xcode Compatibility

The current version has been built and tested with XCode v10.2.