Click2SDX Control App SDK for Android

APIs and SDK integration manual (Beta)

Version 0.0.4

Sep 26, 2016

This page intentionally left blank.

Revision history

Date	Version	Notes
Sep 20, 2016	0.0.3	Initial version (Alpha release)
Sep 26, 2016	0.0.4	Beta release

Table of contents

Introduction

The purpose of this document is to provide documentation for the Click2SDX Control App SDK.

Click2SDX Control App SDK is a cross-operator SDK for enabling sponsored data. It allows some entity (a sponsor) to sponsor mobile data usage so the consumed data does not impact the end-user's data plan. The Click2SDX Control App SDK supports multiple operators and provides ability to assign specific operators for app sponsorship.

SDX Quota Manager service is a web based service that provides the ability to manage quotas of data that may be assigned to users of sponsored data.

Scope

This document provides details on how the Click2SDX Control App SDK works and how it should be used by developers to integrate with their Android applications.

Audience

This document is intended for Tata Communications employees and Tata Communications selected third parties that may be developing the Control applications using the Click2SDX Control App SDK.

High-Level Architecture

Click2SDX Control App SDK enables development of Control-Applications to support the use cases described in the following sections:

3rd Party app sponsorship

For this use case, all data traffic for all 3rd party application (or specific set of applications) may be sponsored, typically for some quota limit granted to the subscriber as a gift or reward.

Control-Application sponsorship (This app sponsorship)

For this use case, the Click2SDX Control App SDK enables the Control-Application to provide a platform to render sponsored content such as mobile advertising. There are three possible ways how this use case can be achieved.

This app sponsorship + 3rd party app sponsorship through VPN

In this case, 3rd party applications together with Control-Application are included in VPN Service. And all network traffic is counted against proper 3rd party app sponsorship. It is needed to set devKey3rdPartyApp and not the devKeyThisApp during initialization. There is no need to use SDX.startThisAppSponsorship() call as the sponsorship of Control-Application will start automatically if there is an active sponsorship.

This app sponsorship (outside VPN) + 3rd party app sponsorship (through VPN)

In this case only 3rd party applications are included in VPN Service. However Control-application content will be sponsored separately by different instance of Click2SDX SDK. To achieve this it is needed to set devKey3rdPartyApp together with devKeyThisApp. These should be distinct. To actually start separate session of Click2SDX to sponsor Control-Application content, SDX.startThisAppSponsorship() method needs to be called. It must be called from different Android process than the Click2SDX Control App SDK was initialized in.

Only This app sponsorship

In this case only Control-Application content will be sponsored. To achieve this it is needed to set devKeyThisApp. To actually start Click2SDX SDK session to sponsor Control-Application content, SDX.startThisAppSponsorship() method needs to be called.

To support the above use cases. The Control-Application is built using a per app VPN Service which allows it to intercept all mobile device network data or only network data associated with specific applications. The SDK provides an API to the Control-Application to allow the Control-Application to start/stop sponsorships and manage the quota.
In addition, the SDK provides notifications and events to the Control-Application. The SDK provides a set of APIs to manage these functions for the application.

Architecture Diagram

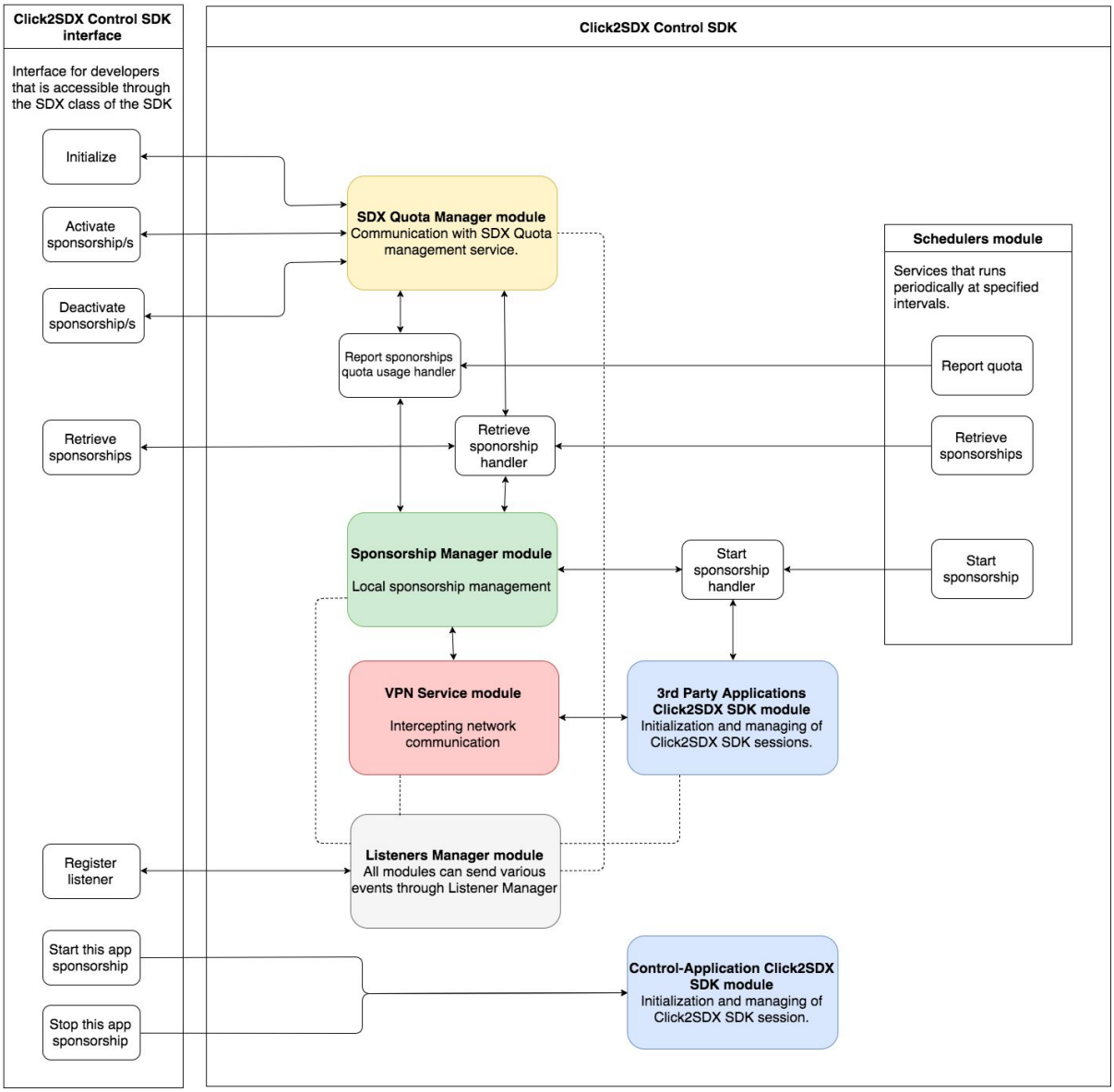

Figure 1. High-Level architecture diagram

Diagram flows

Basicly the flow represents communication between modules. That is implemented as direct calling of methods of specific module or by reacting to various events. Direction of the flows shows if there is two-way communication between modules or not. Dashed lines represents events that are send to Listener Manager module, that can distribute them to the Control-Application.

Initialize

This is triggered by calling SDX.initialize() method. This will basically instantiate all other modules. During initialization SDK communicates with SDX Quota Manager service to register and login user. It also triggers retrieval of sponsorships with help of Retrieve sponsorship handler from SDX Quota Manager service. Result is passed back to provided callback - this stands for most of the SDK interface calls.

Activate sponsorship/s

This is triggered by calling SDX.activateSponsorship() and it activates sponsorship within SDX Quota Manager Service.

Deactivate sponsorship/s

This is triggered by calling SDX.deactivateSponsorship() and it deactivates sponsorship within SDX Quota Manager Service.

Retrieve sponsorships

This is triggered by SDX.retrieveSponsorships(). It retrieves sponsorships from SDX Quota Manager service with help of of Retrieve sponsorship handler. Retrieved sponsorships are processed by Sponsorship Manager module and passed as result back.

Register listener

Various listeners can be registered with help of Listener Manager by calling SDX.register{NAME_OF_THE_LISTENER}(). Listeners are triggered when various events occurs. All other modules can send events to Listener Manager (this is represented by dashed line).

Start/Stop this app sponsorship

This is triggered by calling SDX.startThisAppSponsorship()/SDX.stopThisAppSponsorship(). Control-Application Click2SDX module is then instructed to start/stop session of Click2SDX SDK.

Modules

Click2SDX Control App SDK consists of several modules, that communicate with each other within the SDK. Applications access the functionality of the Click2SDX Control App SDK through its interface.

Click2SDX Control App SDK interface

An application can access all the functions of the Click2SDX Control App SDK through a single interface.

Application can retrieve sponsorships from SDX Quota Manager. After sponsorships are retrieved, they can be activated/deactivated.

Application can register various listeners to be notified when certain events occurs.

Schedulers module

Contains services that run in specific time intervals. There are services for periodical retrieving of sponsorships form SDX Quota Manager, reporting quota usage etc...

SDX Quota Manager client module

SDX Quota Manager module manages communication with the SDX Quota Manager service. All requests to the SDX Quota Manager are handled through this module.

Sponsorship Manager module

Sponsorship Manager module handles local sponsorships management and synchronization with data retrieved from the SDX Quota Manager.

VPN Service module

This module manages the network traffic interception. Quota usage is updated for each sponsorship during the flow of network traffic.

Click2SDX SDK module

Click2SDX SDK module is responsible for initialization and managing the Click2SDX sessions.

Listeners Manager module

Each module can send various events to this module. Listeners manager is responsible for delivering those events to the registered listeners.

Click2SDX Control App SDK interface

Applications interact with the Click2SDX Control App SDK through a single interface. This interface is represented by SDX class. SDX class contains several static methods that can be used by developers.

Initialization

The SDX.initialize() method initializes the Click2SDX Control App SDK prior to being used fully. This asynchronous method saves given configuration parameters and then registers and logs in with SDX Quota Manager. The registration with SDX Quota Manager is performed only when it has not been performed before. If the register and login call is successful, scheduler for retrieving sponsorships from SDX Quota Manager is started. Overall result of initialization is passed to the ResultCallback. Once SDK is successfully initialized and at least one activated sponsorship is retrieved, a sponsored session is started automatically. If an error occurs, error code and message are passed to ResultCallback. This applies to all of the api calls that provides ResultCallback as paramater.

A sponsorship is considered valid if the sponsorship is in activated state, and is not fully consumed or expired.

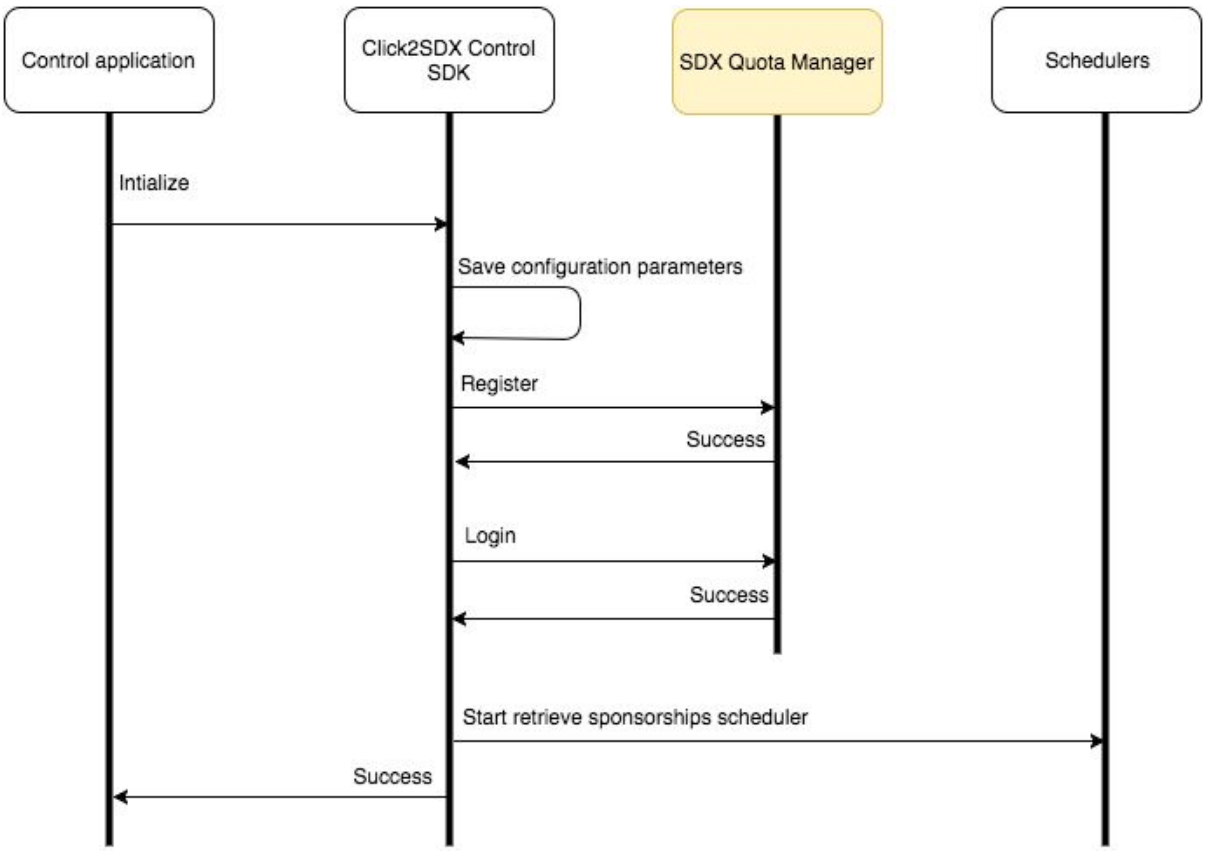

Figure 2. SDX.initialize() method flow

SDX.initialize(Context context, SDXParams params, ResultCallback callback)

This method will initialize Click2SDX Control App SDK with given Android context, configurations parameters and result callback.

SDXParams parameter

SDXParams is builder object that is used for passing all the configuration parameters to Click2SDX Control App SDK.

Below is table with description of each possible parameter.

Parameter	Required	Type	Description
devKey3rdPartyApp	yes if 3rd party application sponsorship is supported	String	Developer key for Click2SDX SDK will be used for network sponsorship of 3rd party applications.
userId	yes	String	User ID for SDX Quota Manager and Click2SDX SDK.
sponsorId3rdPartyApp	no	String	Sponsor ID for Click2SDX SDK the will be used for network sponsorship of 3rd party applications.
devKeyThisApp	yes if Control-Application content sponsorship is supported	String	Developer key for Click2SDX SDK the will be used for sponsorship of Control application content.
sponsorIdThisApp	no	String	Sponsorship ID for Click2SDX SDK the will be used for sponsorship of Control application content.
controlAppId	yes	String	Control application ID as defined by SDX Quota Manager service.
msisdn	yes	String	Phone number of the device.
subscriberType	yes	SubscriberType enum	Subscriber type as defined by SDX Quota Manager. See table below with possible values.

excludedApps	no	List<String>	List of package names of applications that should be excluded from any kind of sponsorship.
notificationParams	no	NotificationParams	Builder object for customizing Click2SDX Control App SDK status notification.
dialogParams	no	DialogParams	Builder object for customizing Click2SDX Control App SDK dialogs.
logLevel	no	LogLevel enum	Sets log level of the SDK. Default value is LogLevel.INFO. See table below with all possible values.
gpsToClick2SDX	no	boolean	If set to true, gps coordinates will be passed to Click2SDX during initialization. Default value is false.
appAuthCode	no	String	If the two factor authentication is performed, appAuthCode needs to be provided.
retrieveSponsorshipsAutomatically	no	boolean	If set to true, sponsorships are retrieved automatically after SDK has been initialized. Default value is true.

Possible subscriberType values:

Value	**Description**
PROPRIETARY_ID	A combination of the userId and controlAppId is used to uniquely identify the recipient
RESTRICTED_MSISDN	A combination of msisdn and controlAppId is used to uniquely identify the recipient.
UNRESTRICTED_MSISDN	Msisdn used regardless of the controlAppId used to receive sponsorships.

Possible logLevel values:

Value	**Description**

DEBUG	All logs will be visible in in Android LogCat.
INFO	All important messages and errors will be visible in Android LogCat.
ERROR	Only errors will be visible in LogCat.

Table for NotificationParams parameter is a nested builder object. Below is the description of its attributes. They are all optional and they are used for customization of status notification provided by Click2SDX SDK.

Name	**Type**	**Description**
sponsoredIconRes	int	Drawable resource that should be used in notification for SPONSORED state.
sponsoredTitle	String	Notification title for SPONSORED state.
sponsoredText	String	Notification text for SPONSORED state.
notSponsoredIconRes	int	Drawable resource for notification icon that is used when the SDK status is not SPONSORED.
notSponsoredTitle	String	Notification title for not sponsored states.
notSponsoredText	String	Notification text for not sponsored states.

Table for DialogParams parameter is a nested builder object. Below is the description of its attributes. They are all optional a they are used for customization of default dialogs that are shown by Click2SDX SDK. These dialogs are:

- Dialog when sponsorship threshold is reached.
- Dialog when sponsorship is consumed.
- Dialog when sponsorship is expired.

Name	**Type**	**Description**
consumedDialogTitle	String	Consumed dialog title.
consumedDialogText	String	Consumed dialog text.
thresholdDialogTitle	String	Threshold dialog title.
thresholdDialogText	String	Threshold dialog text
expiredDialogTitle	String	Expired dialog title.

expiredDialogText	String	Expired dialog text.

For attributes in DialogParams, several placeholders can be used in the provided String. Below is table of those placeholders.

Placeholder	**Description**
${sponsorship}	Will be replaced with sponsorship name.
${remaining}	Will be replaced with sponsorship remaining quota e.g. (100MB)
${threshold}	Will be replace with last reached threshold.
${limit}	Will be replace with sponsorship quota limit e.g. 1GB

Example

If this String is passed as consumedDialogText

```
"All of the ${limit} quota of ${sponsorship} has been consumed"
```

It will be shown as text below in dialog if there is sponsorship with name “Facebook sponsorship” that has quota limit of 1GB and has just been consumed.

```
"All of the 1GB quota of Facebook sponsorship has been consumed"
```

ResultCallback parameter

If you provide ResultCallback, the application will be informed about the result of initialize() method. Same ResultCallback is used in most of the methods provided by SDX interface. ResultCallback has method onResult(Result result), where you can check the result. Result object contains ResultCode that can be ERROR or SUCCESS. In case of ERROR additional message and errorCode are provided for more details.

Possible error codes

Error code	**Reason**
0001	Internal server error.
0002	Out of service of maintenance.
2001	Invalid request.

2004	Invalid subscriber information.
2005	Invalid auth code.
2006	Authorization code required.
2007	Registration required.
2011	Invalid state.

Code sample

The SDK.intialize() should be called on application startup. Android Application's onCreate() method is probably the good place to initialize the SDK. Your custom Android Application class needs to extend SDXApplication class so the SDK can work properly. More info about extending SDXApplication class can be found in setup section.

```
SDXParams sdxParams = new SDXParams();
sdxParams.setDevKey3rdPartyApp("devKey");
sdxParams.setUserIdThisApp("userId");
sdxParams.setSponsorIdThisApp("sponsorId");
sdxParams.setUserId("userId");
sdxParams.setControlAppId("controlAppId");
sdxParams.setSubscriberType(SubscriberType.RESTRICTED_MSISDN);
sdxParams.setMsisdn("phoneNumber");

SDX.initialize(getApplicationContext(), sdxParams, new ResultCallback() {
   @Override
   public void onResult(Result result) {
       if(result.isSuccess()){
           // success
       }else{
           showError();
       }
   }
});
```

If the SDX.initialize() call is successful, sponsorships mayretrieved periodically from SDX Quota Manager service. Retrieved sponsorships are synchronized with locally stored sponsorships.

Retrieving of sponsorships

Sponsorships are retrieved from SDX Quota Manager service. Once SDK is initialized, sponsorships may be retrieved automatically, if enabled, at specific interval specified by SDX Quota Manager.

Manual retrieve of sponsorships can be triggered by the application by invoking the SDX.retreiveSponsorship() method. It is asynchronous and it retrieves sponsorships from SDX Quota Manager and passes them to the provided callback.

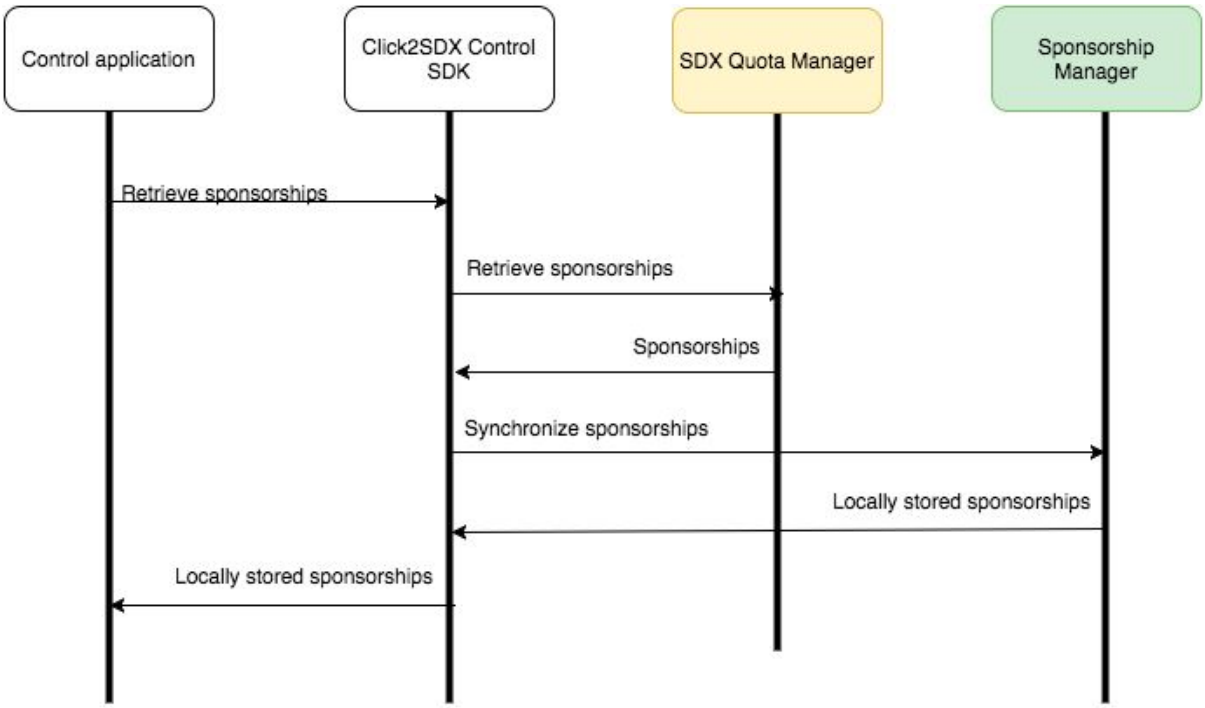

Figure 3. Asynchronous call of SDX.retrieveSponsorships()

SDX.retrieveSponsorships(GetDataCallback<SponsorshipData> callback)

Asynchronous variant with GetDataCallback. Callback has onResult(Result result, List<SponsorshipData> data) method. If the result is SUCCESS you get list of all sponsorships.

SponsorshipData

These are the attributes that can be found in SponsorshipData object.

Attribute	Type	Description
id	String	Identifier of the Sponsorship provided by SDX Quota Manager.
name	String	Name of the sponsorship.
description	String	Description of the sponsorship.
startTime	long	Time from when the sponsorship can be used. Milliseconds since epoch.
expirationTime	long	Time until the sponsorship can be used. Milliseconds since epoch.
totalBytes	long	Total bytes that are available in this sponsorship.
usageBytes	long	Bytes that has been consumed from this sponsorship.
operatorId	String	Id of the operator that this sponsorship is bind to.
operatorName	String	Name of the operator that this sponsorship is bind to.
state	SponsorshipState	State as maintained by SDX Quota Manager.
thresholds	List<Threshold>	List of sponsorship thresholds.
presentationData	String	Custom presentation data send as string.
applications	List<AppInfo>	List of applications that the sponsorship is bind to.

Code sample

```
SDX.retrieveSponsorships(new GetDataCallback<SponsorshipData>() {
   @Override
   public void onResult(Result result, List<SponsorshipData> data) {
```

```
        if(result.isSuccess()){
            showSponsorships(data);
        }else{
            showError();
        }
    }
});
```

De/Activation of sponsorships

Sponsorships can be activated and deactivated in SDX Quota Manager service. There are methods for de/activation single or multiple sponsorships at once. All methods are asynchronous and result is always passed to ResultCallback.

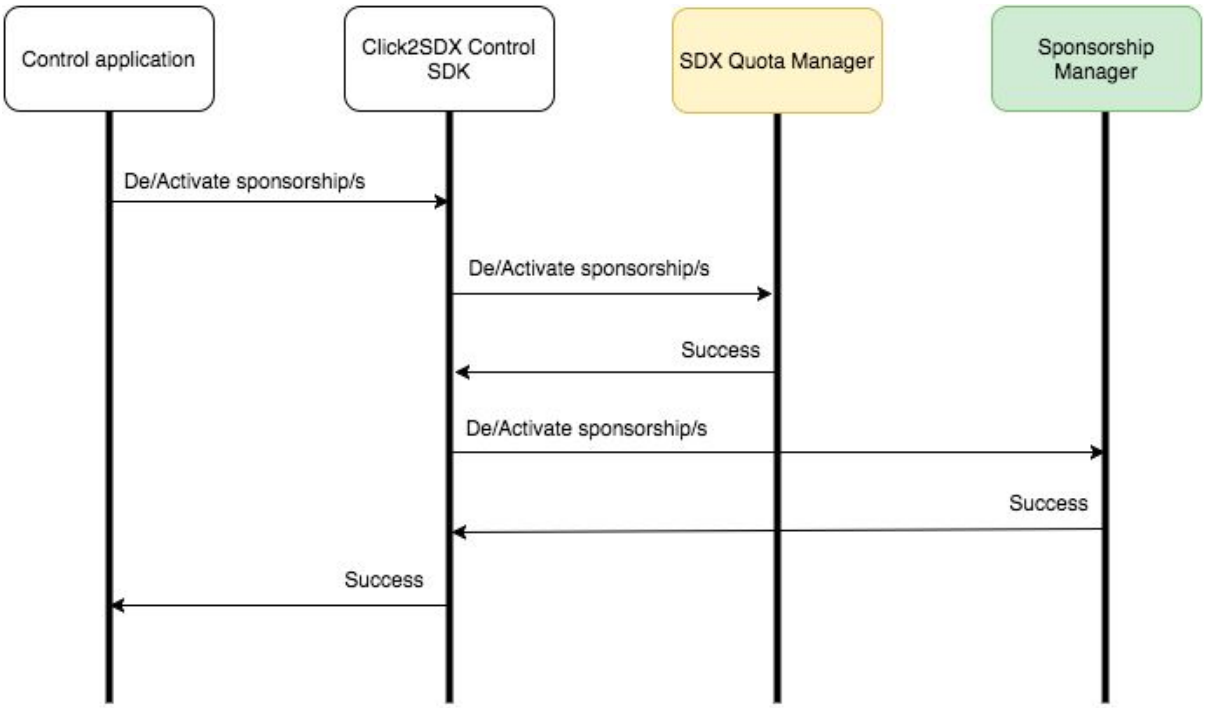

Figure 4. SDX.de/activateSponsorship/s()

SDX.activateSponsorship(String sponsorshipId, ResultCallback callback)

Activates single sponsorship.

SDX.activateSponsorships(List<String> sponsorshipIds, ResultCallback callback)

Activates multiple sponsorships.

SDX.deactivateSponsorship(String sponsorshipId, ResultCallback callback)

Deactivates single sponsorship.

SDX.deactivateSponsorships(List<String> sponsorshipIds, ResultCallback callback)

Deactivates multiple sponsorships.

List<String> sponsorshipIds parameter / String sponsorshipId

Methods for de/activation of sponsorship take sponsorship ID or List of sponsorship IDs as a parameter. Sponsorship ID can be obtained from SponsorshipData object with getSponsorshipId() method.

Code sample

Below you can see code sample for activation of sponsorship/s. Deactivation is identical just replace the activate keyword with deactivate.

```
SDX.activateSponsorship("sponsorshipId", new ResultCallback() {
   @Override
   public void onResult(Result result) {
       if (result.isSuccess()) {
           showSponsorshipActivated();
       } else {
           showError();
       }
   }
});

List<String> sponsorshipIds = new ArrayList<>();
sponsorshipIds.add("sponsorshipId1");
sponsorshipIds.add("sponsorshipId2");
SDX.activateSponsorships(sponsorshipIds, new ResultCallback() {
   @Override
   public void onResult(Result result) {
       if (result.isSuccess()) {
           showSponsorshipsActivated();
```

```
        } else {
            showError();
        }
    }
});
```

Click2SDX Control App SDK state machine

On the picture below you can see all the states of the SDK and transitions between them. States are represented by the colored rectangles. The ovals represents method calls.

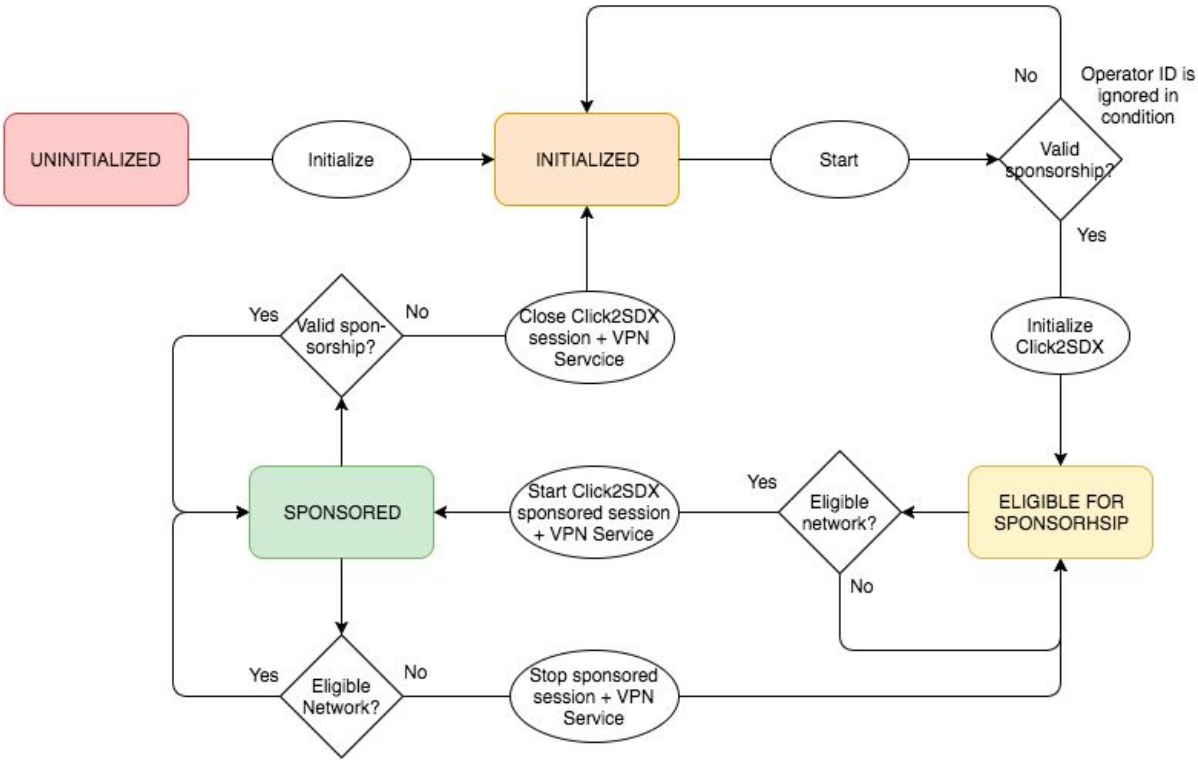

Figure 5. Click2SDX Control App SDK state transitions

Status	Description
UNINITIALIZED	SDK is not initialized. SDK.initialize() needs to be called.
INITIALIZED	Once SDK.initialize() call ends with SUCCESS, SDK moves to this state. In this state SDK is registered and logged with SDX Quota Manager service.
ELIGIBLE_FOR_SPONSORSHIP	Once SDK is INITIALIZED and there is at least one valid sponsorship, Click2SDX session is initialized. If the session is

	initialized successfully, SDK moves to this state.
SPONSORED	If the SDK state is ELIGIBLE_FOR_SPONSORSHIP and device is on eligible network, sponsored session is started within Click2SDX SDK and the VPN Service is started.

You can get actual status of SDK by calling SDX.getStatus() method.

Control-Application sponsorship

Control-Application content can sponsored separately from the 3rd party applications. Or it can be included within 3r party applications sponsorship.
To start separate sponsorship for Control-Application you need to set devKeyThisApp initialization parameter and you need to call SDX.startThisAppSponsorship() method. To stop Control-Application sponsorship you need to call SDX.stopThisAppSponsorship().

This calls needs to performed in another process due to limitations of the Click2SDX SDK. See example below how this can be done. If Control-Application only supports its content sponsorship, there is no need need to created another process.

SDX.startThisAppSponsorship(Context context, SDXParams sdxParams, ResultCallback resultCallback)

Starts the sponsorship of Control-Application content. Needs to be called from a different process than the SDK was initialized.

SDX.stopThisAppSponsorship()

Stops the sponsorship of Control-Application content. Needs to be called in the same process as the SDX.startThisAppSponsorship() was called.

Code sample

New process needs to be created before the sponsorship of Control-Application can be started. This can be achieved with process tag specified in Android application manifest as follows:

```
<activity
   android:name="com.example.ContentActivity"
   android:process=":newProcessName"/>
```

The sponsorship can be then started from ContentActivity's onCreate() method as follows:

```
SDX.startThisAppSponsorship(getContext(), getSdxParams(), new ResultCallback() {
   @Override
   public void onResult(Result result) {
       If (result.isSuccess()){
          // sponsorship started
       }
   }
});
```

And stopped in ContentActivity's onDestroy() method:

```
SDX.stopThisAppSponsorship(getContext());
```

Thresholds

There are three quota thresholds created for each sponsorship by default. They are 50%, 75% and 90% thresholds. Default alert dialog with information that the sponsorships has reached its threshold is shown once for each threshold. Developer can adjust number and values of thresholds by calling synchronous method SDX.setThresholds(). This method overwrites all previously saved thresholds. Thresholds are set as percent of total bytes quota (e.g. 50, 75, 90).

SDX.setThresholds(SponsorshipData sponsorship, Integer… thresholds)

Synchronous method for setting thresholds for sponsorship.

SponsorshipData parameter

Sponsorship for which the thresholds should be set.

Integer… parameter

Array of integers representing percent thresholds. It can be any number between 1 and 99.

Code sample

```
SDX.setThresholds(sponsorship, 70, 80, 90);
```

Listeners

You can register various listeners through the SDX class to be notified when certain events happen (new sponsorship is available, sponsorship is consumed, sponsorship threshold is reached, etc…).
All the methods for listener registration starts with the register…() keyword. To avoid problems with leaks it is important to unregister listeners, when they are no longer needed them. The unregisterListener/s() methods are provided.
To be notified e.g. about sponsorship consumption in an application activity the listener should be registered in Android Activity's onResume() method and it should be unregistered in the onPause() method.

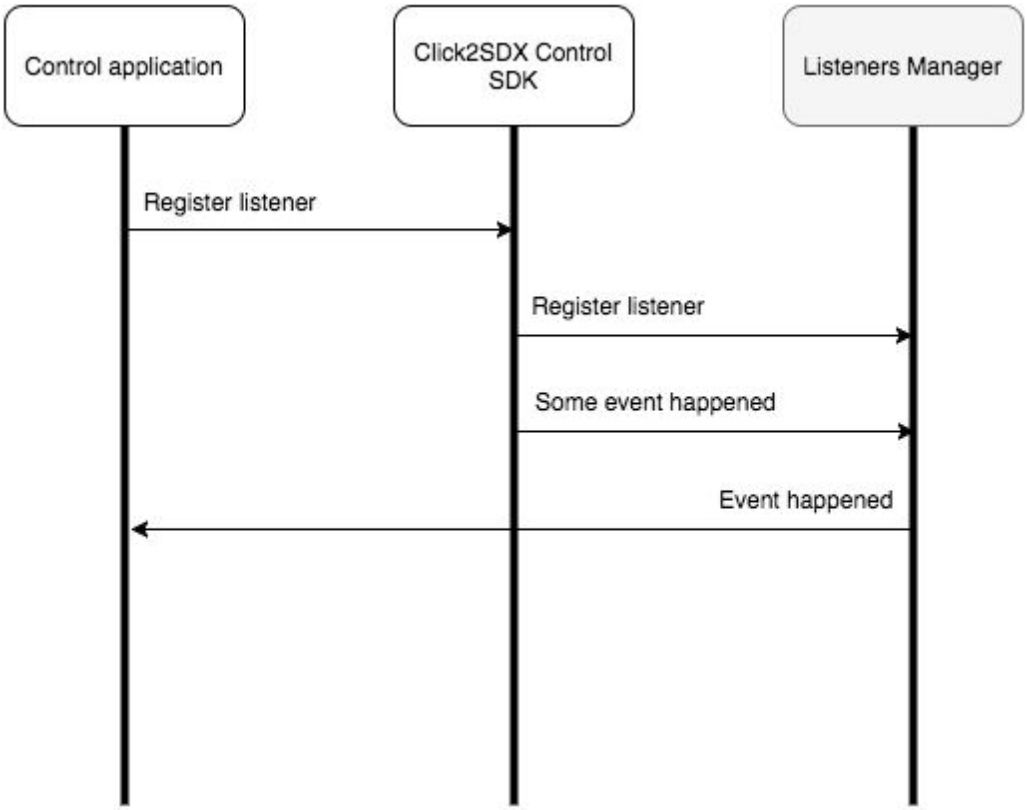

Figure 6. SDX.registerListener() flow

SDX.registerNewSponsorshipListener(Object object, NewSponsorshipListener listener)

Listener will be notified if there is new sponsorship available after each retrieval of sponsorships from SDX Quota Manager.

Object parameter

Object you are registering the listener for. Could be an Activity, Fragment or any other class you wish to register listener for.

NewSponsorshipListener parameter

Listener with onNewSponsorshipsAvailable(List<SponsorshipData> sponsorships) callback. Callback is fired when there is new sponsorship available.

Code sample

```
SDX.registerNewSponsorshipListener(this, new NewSponsorshipListener() {
   @Override
   public void onNewSponsorshipsAvailable(List<SponsorshipData> sponsorships) {
       showNewSponsorships(sponsorships);
   }
});
```

SDX.registerPermissionsRequiredListener(Object object, PermissionsRequieredListener listener)

Listener will be notified when the Click2SDX Control App SDK is required to obtain runtime permission from the user.. It could be Manifest.permission.ACCESS_FINE_LOCATION to get gps position of the device, or Manifest.permission.WRITE_EXTERNAL_STORAGE to allow SDK to write files on device external storage.

PermissionsRequiredListener parameter

Listener with onPermissionsRequired(List<String> permissions) callback. Callback is fired when there is permission required.

Code sample

```
SDX.registerPermissionsRequiredListener(this, new PermissionsRequiredListener() {
   @Override
   public void onPermissionsRequired(List<String> permissions) {
       ActivityCompat.requestPermissions(MainActivity.this, permissions.toArray(new
String[permissions.size()]), PERMISSION_REQUEST_CODE);
   }
});
```

SDX.registerSDXStatusChangeListener(Object object, SDXStatusChangeListener listener)

Listener will be notified if status of Click2SDX Control App SDK is changed. E.g when SDK is moved from UNINITIALIZED state to INITIALIZED state.

SDXStatusChangeListener parameter

Listener with onStatusChange(SDXStatus newStatus, SDXStatus oldStatus) callback. Callback is fired when the status is changed. You get the newStatus together with the previous oldStatus.

Code sample

```
SDX.registerSDXStatusChangeListener(this, new SDXStatusChangeListener() {
   @Override
   public void onStatusChange(SDXStatus newStatus, SDXStatus oldStatus) {
       Log.d(TAG, "new status of Click2SDX Control App SDK: " +
newStatus.toString());
       notificationManager.resolveStausBarIcon(newState);
   }
});
```

SDX.registerSessionStatusChangeListener(Object object, SessionStatusChangeListener listener)

Listener will be notified about the state of Click2SDX session. There are three possible states.

Status	Description
INELIGIBLE	Session is in ineligible state. E.g. device is on WIFI network or other network that is not authorized for sponsorship.
ELIGIBLE	Device is on eligible network.
SPONSORED	Click2SDX sponsored session is running.

SessionStatusChangeListener parameter

Listener with onSessionStatusChange(SessionStatus status) callback. Callback is fired when the Click2SDX session status is changed.

Code sample

```
SDX.registerSessionStatusChangeListener(this, new SessionStatusChangeListener() {
   @Override
   public void onSessionStatusChange(SessionStatus status) {
       Log.d(TAG, "new Click2SDX session status: "+status.toString());
   }
});
```

SDX.registerSponsorshipConsumendListener(Object object, SponsorshipConsumedListener listener)

Listener will be notified if sponsorship is consumed.

SponsorshipConsumedListener parameter

Listener with onSponsorshipConsumed(SponsorshipData sponsorship) callback. Callback is fired when sponsorship is consumed. Consumed sponsorship is passed to callback.

Code sample

```
SDX.registerSponsorshipConsumedListener(this, new SponsorshipConsumedListener() {
   @Override
   public void onSponsorshipConsumed(SponsorshipData sponsorship) {
       showSponsorshipConsumed(sponsorship);
   }
});
```

SDX.registerSponsorshipExpiredListener(Object object, SponsorshipExpiredListener listener)

Listener will be notified if sponsorship is expired.

SponsorshipExpiredListener parameter

Listener with onSponsorshipExpired(SponsorshipData sponsorship) callback. Callback is fired when sponsorship is expired. Expired sponsorship is passed to callback.

Code sample

```
SDX.registerSponsorshipExpiredListener(this, new SponsorshipExpiredListener() {
   @Override
   public void onSponsorshipExpired(SponsorshipData sponsorship) {
       showSponsorshipExpired(sponsorship);
   }
});
```

SDX.registerSponsorshipThresholdReachedListener(Object object, SponsorshipThresholdReachedListener listener)

Listener will be notified if sponsorships threshold is reached.

SponsorshipThresholdReachedListener parameter

Listener with onSponsorshipThresholdReached(SponsorshipData sponsorship, int threshold) callback. Callback is fired when sponsorship threshold is reached. Sponsorship is passed to callback together with reached thhreshold.

Code sample

```
SDX.registerSponsorshipThresholdReachedListener(this, new
SponsorshipThresholdReachedListener() {
   @Override
   public void onSponsorshipThresholdReached(SponsorshipData sponsorship, int
threshold) {
        showThresholdReached(sponsorship,threshold);
   }
});
```

SDX.registerSponsorshipsUpdateListener(Object object, SponsorshipsUpdateListener listener)

Listener will be notified whenever sponsorships in local storage are updated. It can be e.g. when the quota usage is updated during sponsored session, or when sponsorships are retrieved from SDX Quota Manager.

SponsorshipsUpdateListener parameter

Listener with onSponsorshipsUpdated() callback. Callback is fired when sponsorships are updated in local storage.

Code sample

```
SDX.registerSponsorshipsUpdateListener(this, new SponsorshipsUpdateListener() {
   @Override
   public void onSponsorshipsUpdated() {
        showSponsorshipsUpdated();
   }
});
```

SDX.registerSponsorshipUsageListener(Object object, SponsorshipUsageListener listener)

Listener will be notified whenever quota usage is updated for sponsorship during sponsored session.

SponsorshipUsageListener parameter

Listener with onSponsorshipUsageUpdate(SponsorshipData sponsorship) callback. Callback is fired whenever sponsorship usage is updated. Sponsorship is passed to callback.

Code sample

```
SDX.registerSponsorshipUsageListener(this, new SponsorshipUsageListener() {
   @Override
   public void onSponsorshipUsageUpdate(SponsorshipData sponsorship) {
       showUpdatedUsage(sponsorship.getUsageBytes());
   }
});
```

SDX.registerVPNRequiredListener(Object object, VPNRequiredListener listener)

Listener will be notified whenever VPN permission is required by sdk. Intent is passed to onVPNRequired() callback method. This intent needs to be started with startActivityForResult() method to bring up the system VPN Service dialog.

Code sample

```
SDX.registerVPNRequiredListener(this, new VPNRequiredListener() {
   @Override
   public void onVPNRequired(Intent intent) {
       startActivityForResult(intent,REQUEST_CODE);
   }
});
```

SDX.registerVPNForceStoppedListener(Object object, VPNForceStoppedListener listener)

Listener will be notified whenever Android VPNService is stopped by system. This occurs when user removes VPN permission from app or user disconnects VPN using system dialog etc.

VPNForceStoppedListener parameter

Listener with onVPNForceStopped() callback.

Code sample

```
SDX.registerVPNForceStoppedListener(this, new VPNForceStoppedListener() {
   @Override
   public void onVPNForceStopped() {
       showVPNForceStopped();
```

```
    }
});
```

SDX.unregisterListener(Listener listener, Object object)

To avoid leaks it is important to unregister listeners when they are not needed. This call will unregister provided listener for given object. Listener will no longer be notified about events when they occur.

SDX.unregisterListeners(Object object)

This call will unregister all listeners for given object. Listeners will no longer be notified about events when they occur.

Code sample

```
//registration of listener for object
SponsorshipsUpdateListener sponsorshipsUpdateListener = new
SponsorshipsUpdateListener() {
    @Override
    public void onSponsorshipsUpdated() {
        showSponsorshipsUpdated();
    }
};

SDX.registerSponsorshipsUpdateListener(this, sponsorshipsUpdateListener);

// unregistration of listener from object
SDX.unregisterListener(sponsorshipsUpdateListener, this);

// unregistration of all listener from object
SDX.unregisterListeners(this);
```

Miscellaneous

There are several other methods that can be found in SDX class interface.

SDX.getStatus()

Returns status of Click2SDX Control AppSDK.

SDX.isInitialized()

Returns true if the Click2SDX Control App SDK is initialized, false otherwise.

SDX.setStartOnBott(boolean startOnBoot)

Set whether the sponsored session should be started on device boot.

SDX.isStartOnBoot()

Returns whether sdk is configured to start on boot.

Logging

Click2SDX Control App SDK logs various informations at various levels. There are three levels of logging:

- DEBUG
- INFO
- ERROR

Default log level is INFO, however you can specify the log level through logLevel parameter of SDXParams when initializing the SDK.

Click2SDX SDK has separated logging levels. They are DEBUG and ERROR. ERROR is the default level. DEBUG level can be set through debugClick2SDX parameter of SDXParams initializing the SDK.

Setting level of logging to DEBUG is helpful when debugging application, to get rich information about what is going on in the SDK.
INFO level is used for some important events that are happening inside the SDK.
If you set log level to ERROR, only error logs will be visible in the Android LogCat.

You should never set log level to DEBUG for your release builds.

Example

```
SDXParams sdxParams = new SDXParams();
sdxParams.setLogLevel(LogLevel.DEBUG);
sdxParams.setDebugClick2SDX(true);
```

All logs are stored within the SDK and it is possible to retrieve them or clear them. It can be useful for debugging purposes. These are the methods that can be used:

List<SDXLogEntity> SDX.getLogs()

Retrieve all the logs stored within Click2SDX Control App SDK.

SDXLogEntity

These are the attributes that can be found in SDXLogEntity:

Attribute	Type	Description
logLevel	LogLevel enum	Represents level of logging. Possible values: DEBUG, INFO, ERROR
tag	String	Tag of the log. In most cases it represents the class where the log was logged.
message	String	Message of the log.
stackTrace	String	In case the exception is provided to the log, its stack trace is saved in this attribute.
created	long	Time when the log was created.

List<SDXLogEntity> SDX.getLogsByType(LogLevel logLevel)

Retrieves all the logs of given type (log level) stored within Click2SDX Control App SDK.

SDX.clearLogs()

Deletes all the logs store within Click2SDX Control App SDK.

SDX.stopSDK()

This method will uninitialize the Click2SDX SDK. If sponsored session is running, it is stopped. All schedulers are stopped and SDK is moved to UNINITIALIZED state.

Build/Deployment Information

Click2SDX Control App SDK is distributed as Android library project in aar file. Several step are needed to use the SDK. Below is description of how to create an Android application that is using the SDK in Android Studio.

Integration of Click2SDX

First it is needed to integrate Click2SDX SDK into Click2SDX Control App SDK as these are distributed separately.
Cick2SDX Control SDK is provided as *.aar file. You need to change the extension of the *.aar file to *.zip. Once the extension is changed, you can extract the zip file.
Create libs folder inside extracted folder a copy click2sdx-sdk.jar into it together with jni folder containig architecture specific libclick2sdx.so files.

```
extracted_controlsdk/
        /libs
                click2sdx-sdk.jar
        /jni
                /armeabi
                        /libclick2sdx.so
                /armeabi-v7a
                        /libclick2sdx.so
                /x86
                        /libclick2sdx.so

        /other files and folders in extracted_control_sdk
```

Once you have copied Click2SDX SDK files into extracted folder. You can create a new *.zip file from it. Then you have to change the extension of the file back to *.aar.
Now you have *.aar file that can be used in your Android project.

Including aar file

Put controlsdk.aar file in your libs folder in control app module:

Example project structure:

```
/ControlAppProject
        /ControlAppModule
                /libs
                /src
                build.gradle
```

It is important to update build.gradle file of ControlAppModule. You need to add several dependencies and update the repository entry to use libs folder as Flat directory repository.

```
repositories {
   flatDir {
       dirs 'libs'
   }
}

final SUPPORT_VERSION = '24.1.0'
final GPS_VERSION = '9.2.1'

dependencies {
   //sdk dependency
   compile(name: 'controlsdk', ext: 'aar')

   //dependencies required by sdk
   compile "com.android.support:appcompat-v7:${SUPPORT_VERSION}"
   compile "com.android.support:design:${SUPPORT_VERSION}"
   compile "com.android.support:recyclerview-v7:${SUPPORT_VERSION}"
   compile "com.google.android.gms:play-services-base:${GPS_VERSION}"
   compile "com.google.android.gms:play-services-location:${GPS_VERSION}"
   compile 'com.android.support:multidex:1.0.1'
   compile 'com.squareup.retrofit2:retrofit:2.1.0'
   compile 'com.squareup.okhttp3:okhttp:3.4.1'
   compile 'com.squareup.okhttp3:logging-interceptor:3.4.1'
   compile 'com.squareup.retrofit2:converter-gson:2.1.0'
   compile('net.zetetic:android-database-sqlcipher:3.5.3@aar')
   compile 'com.scottyab:secure-preferences-lib:0.1.4'

   //other dependencies used by ControlAppModule

}
```

Alternatively you can import aar file as a module of your project in Android Studio.
File -> New -> New Module -> Import .JAR/.AAR Package

In this approach you don't need to update the repository entry of build.gradle, but you still need to include all the required dependencies with one exception:

```
compile(name: 'controlsdk', ext: 'aar')
```

Is replaced with:

```
compile project(':controlsdk')
```

Extending SDXApplication

Your application needs to extend SDXApplication so the Click2SDX Control App SDK can work properly.

```
public class ControlAppApplication extends SDXApplication {

}
```

The AndroidManifest.xml <application> tag needs to be modified to reflect custom Android Application class:

```
<application
   android:name="com.controlapp.application.ControlAppApplication
   android:icon="@mipmap/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme">
</application>
```

Using the SDK

Once you have imported aar file of the Click2SDX SDK and extended the SDXApplication correctly, you can initialize the SDK through the SDX.initialize() method and start using it as it is described above.

Allowing VPN

One more thing is needed before sponsored session can be established. User needs to allow that the Control application can use VPN Service.
Whether VPN Service is allowed can be checked by calling VPNService.prepare(getContext()). If this method returns null, VPN is allowed, otherwise the Android Intent is returned. If the returned intent is passed to startActivityForResult(), the dialog that allows user of the app to enable VPN Service is shown. VPNRequiredListener can be registered to get notified whenever VPN Service permission is required.

```
Intent intent = VpnService.prepare(this);
if (intent != null) {
   // VPN Service not allowed
   startActivityForResult(intent, REQUEST_CODE);
}else{
   // VPN Service allowed
}
```

Features requiring additional actions

Passing gps coordinates to Click2SDX SDK

If you want to pass gps coordinates to the Click2SDX SDK you need to set gpsToClick2SDX initialization parameter to true. Since Android API level 23, runtime permission (Manifest.permission.ACCESS_FINE_LOCATION) is required for getting the the gps coordinates.
You can request the permission on startup of Control-Application or anywhere you want, or you can request when it is actually needed by SDK.

You can register listener for permissions requested by SDK with SDX.registerPermissionsRequiredListener() which has a callback with requested permissions.

More about requesting runtime permissions can be found here:
https://developer.android.com/training/permissions/requesting.html

Uploading Logs and Session history file

Session history and debug logs can be uploaded to Tata servers for debugging purposes. Creating those files requires runtime permission (Manifest.permission.WRITE_EXTERNAL_STORAGE).
You can request the permission in Control-Application at any time you want or when it is actually needed by SDK. There is a PermissionsRequiredListener for that. You can register it with SDX.registerPermissionsRequiredListener() and get notification when the permission is required by SDK.

More about requesting runtime permissions can be found here:
https://developer.android.com/training/permissions/requesting.html

Building sample applications

In the Click2SDX Control App SDK zip file, there are an example applications (examplegiftapp & examplerewardapp) which demonstrates how the SDK should be used for a specific use case.

Each of the example applications is a complete Android project that can be opened and built in Android Studio.

If want to run the example application on device or emulator you need to update initialization parameters to reflect your needs.

There is com.click2sdx.controlsdk.exampleapp.android.manager.InitManager class that is responsible for initializing the Click2SDX Control App SDK. There is getSDXParams() methods where the actual SDXParams is created. You can adjust initialization parameters as you want. It is needed to change the controlAppId, devKey3rdPartyApp and devKeyThisApp with your own values. Look for a //TODO: comment in the InitManager class.

examplegiftapp

Examplegiftapp demonstrates how the SDK is to be used for the phone gift use case. This application itself does not provide any sponsored content.

Application automatically retrieves the Gift and presents it (them) to the user. If there is new Gift, notification is shown. User can activate/deactivate available Gifts. When any Gift is activated sponsored session will be started whenever the device is on eligible network.

examplerewardapp

Examplereward demonstrates how the SDK is to be used for the rewards case. This application provides its own content that would be sponsored.

Application automatically retrieves the Rewards and presents it (them) to the user. If there is new Reward, notification is shown. User can activate/deactivate available Reward. When any Reward is activated sponsored session will be started whenever the device is on eligible network. Sponsored content in the form of web view is presented by the Control-Application itself.

Libraries used by Click2SDX Control App SDK and their licences

Retrofit

http://square.github.io/retrofit/
http://square.github.io/retrofit/#license

OkHttp

http://square.github.io/okhttp/

http://square.github.io/okhttp/#license

SQLCipher

https://www.zetetic.net/sqlcipher/
https://www.zetetic.net/sqlcipher/license/

Secure-preferences

https://github.com/scottyab/secure-preferences
Apache License, Version 2.0

LocalVPN

https://github.com/hexene/LocalVPN
Apache License, Version 2.0