Contents

Version history

Version	Date	Description	Resp.
A	13/06-2016	First release	TH
B	21/12-2016	Review/Comments and feedback from SEB	TH/BJ

1 Overview

The dynamic re-engagement is available from CDO 2.3. The feature allows a publisher to dynamically re-engage his users and therefore increase user engagement by deep-linking directly into his app from every Caller ID screen. This can extend the lifetime value of the app.

This is achieved through the Calldorado website where an Engagement field needs to be set up being of the type "Dynamic", along with a field name and a corresponding deeplink URL. The Engagement field is simply an action on the after call screen that contains an image along with a motivational message that enables the user to tap on the action. Whenever a user taps on the action, Calldorado launches the hosting app with a special parameter (the deeplinkUrl, which is explained below) which the app then can utilize to run custom logic, e.g. taking the user to a special place within the app or launch a website.

The publisher is able to dynamically re-engage his users at runtime through his own code. He needs to use this line within his app when he wants to setup a dynamic re-engagement field:

Calldorado.setupDynamicReEngagementField (<Context context>, <ReEngagementField reEngagementField>);

The context can be of any (e.g. Application, Activity, Service and so on).
The "reEngagementField" object consists of the following attributes:

- **fieldname**
  The fieldName represents the re-engagement id, which will be used in order to show the reengagement field on the after call screen. This name should also be set on the Calldorado website, otherwise the reengagement field won't be shown.
- **deeplinkUrl**
  The deeplinkUrl will be sent back to the publisher app's launcher activity, when the user taps on the reengagement field on the after call screen. The publisher is then able to react upon the deeplinkUrl.
- **Message**
  The message is shown on the after call screen reengagement field.
- **imageId**
  The imageId determines what internal image that should be shown; can be between 1 - 9 (if not set, the default reengagement image will be shown – The icons can be seen on my.calldorado for your app)
- **imageByteArray**
  The byte array will be a custom image set by the publisher (if a custom image is not wanted, "null" can be passed in as parameter.
- **startDate**
  The start date for the reengagement field (default current date - The SDK will validate whether current date is within or equal to <startDate> and <endDate> and if not, the reengagement field will not be shown)
- **endDate**
  The end date/expiry date for the reengagement field (default the maximum date possible –

if current date is not within or equal to <endDate>, the reengagement field will not be shown)

2 Class doc implementation

```
/**
 * ReEngagementField class which is used to construct the reengagement field
 */
public static class ReEngagementField {

    private String fieldName;
    private String deeplinkUrl;
    private String message;
    private int imageId;
    private byte[] imageByteArray;
    private long startDate;
    private long endDate;

    public ReEngagementField(String fieldName, String deeplinkUrl, String message) {
        this(fieldName, deeplinkUrl, message, 0, Calendar.getInstance().getTime().getTime(), Long.MAX_VALUE - 1);
    }

    public ReEngagementField(String fieldName, String deeplinkUrl, String message, byte[] imageByteArray) {
        this(fieldName, deeplinkUrl, message, imageByteArray, Calendar.getInstance().getTime().getTime(), Long.MAX_VALUE - 1);
    }

    public ReEngagementField(String fieldName, String deeplinkUrl, String message, int imageId, long startDate, long endDate) {
        this.fieldName = fieldName;
        this.deeplinkUrl = deeplinkUrl;
        this.message = message;
        this.imageId = imageId;
        this.startDate = startDate;
        this.endDate = endDate;
    }

    public ReEngagementField(String fieldName, String deeplinkUrl, String message, byte[] imageByteArray, long startDate, long endDate) {
        this.fieldName = fieldName;
        this.deeplinkUrl = deeplinkUrl;
        this.message = message;
        this.imageByteArray = imageByteArray;
        this.startDate = startDate;
        this.endDate = endDate;
    }
```

As shown on the image, the publisher is able to construct a reengagement field using one of these four constructors. **fieldName**, **deeplinkUrl** and **message** must be set when constructing the reengagement field.
For each attribute, a getter and a setter is provided.

3 Overall flow and examples

1. As the first thing, the publisher has to setup the reengagement field on the website with a ***fieldName*** and mark it as "*Dynamic*."
2. Publisher sets up a reengagement field in his code and calls *Calldorado.setupDynamicReEngagementField (<Context>, <reEngagementField>);*
3. This Reengagement is broadcasted into Calldorado and stored locally in a SQLite database.
4. When entering the after call screen, we fetch the reengagement field by combining the locally stored client reengagement with the reengagement setup fetched from the server - If the reengagement ***fieldName*** fetched from the client is recognized using the server reengagement ***fieldName***, the reengagement is shown on the after call, otherwise the reengagement field will not be shown.
5. When the user taps on the field, the user is navigated back to his launcher activity in his app, in which he can get the intent data containing the ***deeplinkUrl***.

Example:

1. The Publisher logs in to his account on my.calldorado, navigates to the app he wants to create a dynamic reengagement for and creates a dynamic reengagement by providing a ***fieldName –*** The fieldName should be exactly the same used in the app in order to have it shown on the after call screen.

2. Publisher sets up a reengagement field in his code and make a call from e.g. a button click: Below is just an example with values provided. The values can be anything set by the Publisher. Please make sure that the ***fieldName*** is equal to the one created on my.calldorado. The ***deeplink*** Url and the ***message*** can be text messages that suites you.

```
if (btnText.equals("1")) {
    phoneNumberEt.setText(phoneNumberEt.getText().toString() + 1);


    Calldorado.ReEngagementField reEngagementField = new Calldorado.ReEngagementField("TestSpeeddialerEngagement", "http://play.google.com/store/apps/details?id=" + getPackageName()
            , "Visit speeddialer on google play and speed up your calls");
    Calldorado.setupDynamicReEngagementField(this, reEngagementField);
```

The client calls the ***Calldorado.setupDynamicReEngagementField(context, reEngagementField)*** method passing the actual context, in this case an activity, and a reengagement field object with the following attributes:

- *TestSpeeddialerEngagement* (***fieldname***) – should be the same name on the website.
- *Google play url* (***deeplinkUrl***) – what the client wants to do when pressing the Reengagement field on the after call screen. In this case, it is a Google Play URL that shows my app on Google Play.)
- *Visit speedialer on…* (***message***) - The actual message shown in the reengagement field on the after call screen.

3. When the method is executed, the reengagement field attributes will be broadcasted into Calldorado and stored locally.

4. Finally, when the client enters the after call screen, we fetch the reengagement we want to show using the values set in publisher's app. If a valid reengagement can be fetched, it is shown on the after call screen.

5. Whenever a user taps on the reengagement field on the after call screen, the user is navigated back to the publisher's launcher activity inside his app. Here, the user is able to "listen" to the received intent and do the appropriate action based on the ***deeplinkUrl*** provided. It is done by doing following:

```
Intent i = getIntent();
try {
    Uri data = i.getData();
    String path = data.getPath();
    Log.d(TAG, "ReEngagementField path: " + path);

    if (path.equals("...")) {
        Toast.makeText(this, "Path = " + path, Toast.LENGTH_LONG).show();
    } else if (path.equals("....")) {
        Toast.makeText(this, "Path = " + path, Toast.LENGTH_LONG).show();
    }
    //add more cases if multiple links into app.

} catch (NullPointerException e) {
    // Catch if no path data is send
}
```

*Inside the onCreate in the launcher activity, you simply use this code and do the appropriate action depending on the actual path you are receiving. The path is the **deeplinkUrl** that is provided in the dynamically created ReEngagement.*

4 Dynamic Re-engagement event

In case the Publisher wants to get notified whenever the reengagement field is shown on the after call screen, he can implement a Broadcast Receiver that listens to the event. This might be very useful if the publisher wants to update the reengagement field dynamically after a call has been made. The Publisher can simply react to this event callback.

The procedure is as follows:

1. The publisher has to create a class which extends BroadcastReceiver that listens to the following action: "**com.calldorado.android.intent.DYNAMIC_RE_ENGAGEMENT_SHOWN**"
2. The Publisher has to register this Broadcast Receiver in his manifest.

The following example illustrates what needs to be done:
I have called my Broadcast Receiver for "DynamicReEngagementReceiver", however, you can call it whatever you want to.

```
public class DynamicReEngagementReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals("com.calldorado.android.intent.DYNAMIC_RE_ENGAGEMENT_SHOWN")) {

            Bundle bundle = intent.getExtras();
            String reEngagementName = bundle.getString("reEngagementName");

            Calldorado.ReEngagementField reEngagementField =
                    new Calldorado.ReEngagementField(reEngagementName, "deeplink", "message");
            Calldorado.setupDynamicReEngagementField(context, reEngagementField);
        }
    }
}
```

The above example shows a custom class called “DynamicReEngagementReceiver” that listens to the required action. In order to update the reengagement field, it’s important to know that you MUST get the reEngagementName from the intent bundle and use that as part of your new dynamicReEngagement field setup.

Don’t forget to add the created Broadcast Receiver in your manifest.

```
<receiver android:name=".DynamicReEngagementReceiver">
    <intent-filter>
        <action android:name="com.calldorado.android.intent.DYNAMIC_RE_ENGAGEMENT_SHOWN"></action>
    </intent-filter>
</receiver>
```