**Twilio Contact SDK Development Notes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **Revision Summary** | | | | |
| **Rev** | **Description of changes** | **Changes by:** | **Approval by:** | **Date** |
| 1.0 | Initial Draft | *John Yang* |  | 08/09/2021 |
|  |  |  |  |  |
|  |  |  |  |  |

**Purpose of Document:**

This document is intended as a development note for the development of the Contact SDK.

This document talks about the what I have done, choice of design for each component of the project, improvements made to original architecture, testing strategies, and some tasks that need to be done as future improvement.

**What is Contact SDK?**

This project is a minimalistic implementation of a cross-platform mobile SDK to manage “Twilio Contacts”. We have a cloud-based distributed system to manage phone contacts - first name, last name, and phone number - and this project servers as the mobile client libraries to access those contacts in a cross-platform way.

On the client-side, we have a common C++ library that will be used by both the iOS and Android SDKs, this project is scoped to the Android platform, with 3 major components:

1. C++ Contacts SDK

2. Android Contacts SDK - This SDK use the C++ SDK

3. Sample Android application that demonstrates the SDK capabilities.

**Code Repo:**

Github repo: <https://github.com/JohnYang524/AndroidContactSDK>

(Note: This repo is private. Please let me know your email address if you want access to it)

**What I have done:**

APIs:

I’ve updated the SDK to provide APIs for the followings:

1. List contacts - return a list of known contacts
2. Get list all of user’s contacts that were updated/created after a certain timestamp
3. Add a new contact – asynchronously notify the caller of success/failure
4. Notify user of an updated contact event.
5. Get timestamp of last DB server data update.

C++ Library:

* Created **EventListener** class to handle callbacks in case of Contact events
  + EventListener(jweak pJObject, jmethodID pID); // jclass and jmethodID
  + I created a vector<EventListener \*> to store all listeners (converted to EventListener objects) passed in from the mobile SDK.
  + During contact-updated event, I will try to notify all listeners.
* Created the following JNI native methods:
  + ***void JNICALL Java\_com\_contacts\_Contacts\_nativeAttachListener(JNIEnv \*env, jobject jclass, jobject listener);***
    - Called when a listener is attached from the application side to attach listener to C++ code.
    - Here I am also caching a JavaVM instance to get JNIEnv instance when needed.
    - Also creating EventListener object and saving it in EventListener vector
  + ***jstring JNICALL Java\_com\_contacts\_Contacts\_nativeGetContactList(JNIEnv \*env, jobject jclass);***
    - This is the call to get a list of all known contacts.
    - For testing, I am returning the test data using getContactsList() from contacts.cpp
  + ***jstring JNICALL Java\_com\_contacts\_Contacts\_nativeGetUpdatedContactListAfter(JNIEnv \*env, jobject jclass, jstring timestamp);***
    - I created this method here as an effort to potentially improve server request efficiency.
    - **Instead of querying for all contact data every single time, I am only querying updated/newly-created data that were updated after a certain timestamp.**
    - On the application side, we will store a local DB update timestamp, and before querying for contact data, we will query for the server last-updated timestamp. Then, we will compare the 2 timestamp and determine if we will need a server call to update the list
    - To implement this logic, I created native method to get server update timestamp, and used Room Database to store data on the app side.
    - For testing, I am still returning the whole data list every time.
  + ***jstring JNICALL Java\_com\_contacts\_Contacts\_nativeGetLastUpdateTime(JNIEnv \*env, jobject jclass);***
    - Returns timestamp of last server data update
  + ***void JNICALL Java\_com\_contacts\_Contacts\_nativeAddNewContact(JNIEnv \*env, jobject jclass, jstring contactData, jobject listener);***
    - Accept new user data, sends to server, and call back to application with server response.
    - Call addNewContact(newContactData) in contacts.cpp to make server calls and receive a server response
    - Then call back to the listener passed in with the server response
  + ***void JNICALL Java\_com\_contacts\_Contacts\_nativeSimulateContactUpdateEvent()***
    - This is my test method created to simulate a server contact update triggered from the application side
    - Under the demo app actionbar menu, there are 2 test options: “Simulate a Server Update” and “Restore contact data”. This method will be called when “Simulate a Server Update” is clicked.
    - This will call onContactUpdated(jstring) to test callback functions.
  + ***void onContactUpdated(\_jstring \*updatedContact\_)***
    - Calls back to java listener when server data is updated  
       \* Step1: Get current JNIEvn from cached JVM

\* Step2: Notify all listeners stored in vector, even though in our test case there is only 1 listener.

\* Step3: Detach current thread

* Created member functions in **contacts.h / contacts.cpp** and test data:  
   *static std::string getContactsList();*

*static std::string getUpdatedContactsListAfter(std::string timestamp); // Returns only contact records that got updated after timestamp*

*static std::string getLastUpdatedTime(); // last updated time in DB server*

*static int addNewContact(std::string newContact); // add new contact, returns server response success/failure*

*static void onContactDataUpdated(std::string updatedContact);// Call back to upper SDK for update events*

*static std::string getTestDataForCallbackTest();*

*static void addEventListener(EventListener \*listener);*

Android SDK:

*Contacts:* Singleton class for making Contacts API calls.

* This class contains method to make native calls to C++ library.
* It holds a ContactManager instance where business logic and the callback interface is kept.

*ContactsManager:* Class for managing contact data update flow and keeping business logic.

* Definition for event listener to handle callbacks  
  *public interface ContactEventListener {*

*void onContactListLoaded();*

*void onNewContactAdded(int responseCode);*

*void onContactUpdated(String newContactData);*

*}*

* This class holds a list of List<Contact> with which we feed the RecyclerView in the app.
* This class holds methods for managing contact data and business logic for updating Room DB. (Some key methods listed below)
  + ***List<Contact> getContactList()***
    - This method returns the list of Contact data stored locally
  + ***Void syncContactList()***
    - Based on Room DB last sync timestamp and DB server last update timestamp, this method updates local contactList either from data in Room DB, or from Web service call responses.
    - **Steps took in this method to update contactList**:
      * Step 1: Retrieve last update time in server
      * Step 2: Based on last sync timestamp, the list is updated either from Room DB, or from Web service call responses from C++ SDK
      * Step 3: If list is updated with server data, update local DB and update lastUpdateTimestamp
      * Step 4: Update local contactList
  + ***void addNewContact(Contact contact, Context context)***
    - Add a new contact record.
    - Send contact json string to C++ library
    - Send a listener to receive callback from C++ library
    - Update local DB
  + ***void onContactDataUpdate(String newContactData, Context context)***
    - Called when contact data is updated in server.
    - Update local DB and local list.
    - Notify UI

*Util:* Class for keeping utility methods. E.g. JSON parser, SharedPref, etc.

*ContactDatabase*: Room database for saving contacts data.

* Singleton class because constructing a Room DB instance is heavy.

*ContactDAO:* Room database Contact DAO

Android SDK Test:

* *ContactsTest:* Android instrumentation tests
* *ContactsUnitTest*: Unit tests
* *TestData:* Sample test data

Demo Application(UI):

Android application to demonstrate the capabilities of the Contacts SDK you have implemented.

This application includes one Acitivity and two Fragments. The transition between 2 Fragments is handled with Android Navigation components.

ContactsActivity: Main activity that holds <fragment> in its layout.

ContactListFragment: Fragment to display a list of Contacts

* This fragment holds a RecyclerView to display contact data list with help of ContactListAdapter
* Logic for fetching Contact Data to feed list.
  + If there is data in local contactList -> show them in UI right away
  + At the same time, check for server updates based on timestamp
  + If there is new updates in server, fetch it, update DB and update UI (handled through SDK)
* A ContactEventListener is held in this class to listen to update event.

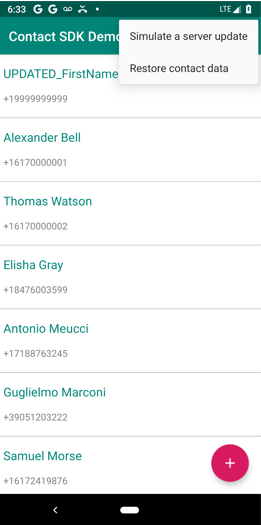
CreateNewContactFragment: Fragment to create a new contact record.

Contact data

* Added an “ID” field for each contact record to better align with real cases and usage of Room Database.
* For simulating DB sequential id, I created a utility method *Util. getNextID()* to simulate auto-incrementing function.

**Testing Contact-Updated Event Callbacks:**

For testing the callback function in the event of server data updated, I added an option menu item “Simulate a Server Update” under option menu (Screenshot Below).



Clicking the “Simulate a server update” button will call the nativeSimulateContactUpdateEvent. In the C++ library, we will simulate a data update and call native method onContactUpdated(). This method will then update all stored EventListener with the updated data. In our case, our listener created in ContactListFragment will be notified. We will raise a toast if the listener receives the callback, and then update Room DB and UI accordingly.

Clicking the “Restore contact data” will restore the local data with the original test data in C++. This is done by deleting all data in DB and do a data sync with C++ library.

**Future tasks:**

1. Demo app - Implement a ViewModel and use LiveData to observe contact data change
2. Android SDK – Implement a custom UI (RecyclerView) that handles some of the contact data update events from within the SDK
3. C++ SDK – Implement more meaningful APIs.
4. More test cases.
5. Automatic testing with Appium

**Known Issue:**

We have one known issue. When testing event callback by clicking the “Simulate a server update” menu item, I’ve seen some errors from the JNI side:

“JNI DETECTED ERROR IN APPLICATION: JNI CallVoidMethod called with pending exception: Fragment not attached to a context”.

Step to reproduce: Rotate the screen -> then click the “Simulate a server update” button

Potential root cause: Fragment was detached and re-attached to the activity on orientation change. The original Fragment object might have been detached or recycled. When JNI was trying to call back with that recycled object, it threw an error.

Potential fix:

1. Re-attatch listener if we detect original Fragment or listener is detached.
2. On the JNI side, check to see if object is still attached to a context before making the call
3. On the app side, refactor demo app to use ViewModel to handle data and survive orientation change.