Lesson 1:

Setup Project, Use MTI Lib and Start MapTrip

Introduction

This lesson is the initial start into using the MapTrip Interface.

During this lesson you will
- prepare the sample project
- include the MTI API into your project
- start MapTrip as a server
- implement MTI Callback interfaces for Basic functions
- initialize the MTI API for further function calls

Prerequisites

You should have experience working with Java programming language and the Android Studio development environment.

You should also be able to work with git and github respectively.

The execution of the sample app which you will create during the lessons requires the installation of the MapTrip App at the same device.

The lessons of this tutorial build on each other. The result is a simple app that provides the essential functions of the MTI interface.

For a quick start and to be able to work with the individual lessons easily, we recommend using the app that we also provide in our Git account. This app is something like a frame for your learning steps.

Using MTI

Important to understand how the MTI API works is that this library is something like a communication layer between your app and the MapTrip Navigation App. MTI enables your app sending commands to MapTrip and reacting to MapTrips responses.

In addition, MTI notifies your app when status changes have occured in MapTrip, for example during navigation.

To use MTI there are some less steps required. First part is including the MTI library into your software. The next steps are initializing the API and making sure that the App MapTrip is running.

Start with Project

When you begin learning with this tutorial at first (and only for one time) you should clone the basic tutorial app.

Cloning the MTI Sample App

- SSH: git@github.com:MapTrip-Navigation/MTI-Tutorial.git
- HTTPS: https://github.com/MapTrip-Navigation/MTI-Tutorial.git
- GitHub CLI: gh repo clone MapTrip-Navigation/MTI-Tutorial

Open the MTI Sample App Project with your Android Studio

Click at File menu and select the folder ***app*** of the downloaded project files.

HowTo: Using MTI API Library

After opening the project at first you have to integrate the lib. Of course there are several ways, the next steps is one of them.

- In Android Studio open the file build.gradle (Module: app), declare the infoware repository and complete the dependencies:

```
repositories {
    maven {
        url "http://artifactory.infoware.de:8080/artifactory/libs-gradle/"
    }
}

dependencies {
      implementation „de.infoware:mti:8.5.4“
      ...
      ...
}
```

Don’t make changes to the other already existing lines in this part.

After your changes are made synchronize your project with the Android Studio IDE. Then MTI is ready to use in your project.

HowTo: Start MapTrip as Server App

Before you can take control over MapTrip the App must be started. At this point we assume that MapTrip is installed and runnable. To see if our implementation works, MapTrip shouldn’t already run.

Starting another App is done by *Intents*. Intents are not part of MTI but an Android specific mechanism which enables the interacting between Apps or GUI parts of Apps.

1. Create a new class in package lessons: **Lesson1_Initialize.java**
2. Extend the new class from class **Lesson.java**
    - Confirm the Android Studio proposal for implementing the abstract methods.
    - Confirm the proposal for creating the super constructor

```
public class Lesson1_Initialize extends Lesson {

    public Lesson1_Initialize(int functionId, String buttonCaption,
Fragment fragment) {
        super(functionId, buttonCaption, fragment);
    }

    @Override
    public void doSomething() {

    }

}
```

3. Add the first button to the button list of the GUI

   The class **HomeFragment.java** stores a list of Lessons (extensions of class Lesson). We extend this list with our before created class **Lesson1_Initialize.java**:

```
private ArrayList getLessons () {
    ArrayList<Lesson> lessonArrayList = new ArrayList<>();
    int lessonIndex = 0;

    getActivity().getPackageCodePath();

    // Add your first lesson class here
    lessonArrayList.add(new Lesson1_Initialize(lessonIndex++,"Start
MapTrip And Init MTI", this));

    return lessonArrayList;
}
```

   Don’t forget to import the class Lesson1_Initialize.

4. **Lesson1_Initialize.java**: override the method doSomething() of base class Lesson

   This method will be called when the button START MAPTRIP AND INIT MTI is clicked. From here we call the below implemented method startMapTrip().

```
@Override
public void doSomething() {
    startMapTrip();
}
```

5. **Lesson1_Initialize.java**: implement the method startMapTrip()

   This method creates an Intent and starts the Activity MapTrip.

```
private boolean startMapTrip() {
    Intent intent =
fragment.getActivity().getPackageManager().getLaunchIntentForPackage("de.i
nfoware.maptrip.navi.license");
    if (null == intent) {
        return false;
    }
```

```
        try {
            fragment.getActivity().startActivity(intent);
        } catch (ActivityNotFoundException eToo) {
             //
        } catch (Exception ex) {
            //
        }
        return true;
    }
```

Don’t forget importing classes.

From now a click at the Button START MAPTRIP AND INIT MTI should start MapTrip and set it to front.

HowTo: Working with MTI Callbacks

There are several options handling the MTI callbacks. The below described solution is only one of many possible variants!

Every app can only register one listener to MTI callbacks. With this implementation different classes can react to incoming callbacks.

The class **MtiListener** implements the callback interface. This listener receives the incoming MTI (MapTrip) messages and forwards them to other classes if necessary. In our case this classes are lessons.
Lessons have to ‚register‘ at this class to get incoming callback informations.

The next steps are preparations of our new listener class. Later we will complete the forwarding mechanism within the different callback methods.

The MTI callback listener

1. Create a package **listener**
2. In the new package create the class **MtiListener.java**
3. Make the new class as an implementation of **ApiListener**
   Let Android Studio create the method implementations for you
4. Create a member which holds registered lessons
   ```
   private ArrayList<Lesson> registeredLessons = new ArrayList<>();
   ```

5. Create a method for registering lessons
```
public void registerLesson(Lesson lesson) {
   registeredLessons.add(lesson);
}
```
6. Providing callback informations to registered classes
   For every callback we implement a counterpart in the class **Lesson**.

Usage of the callback listener

Extend the class **Lesson** for providing the callback listener and for registering lessons.

1. Extend Lesson from MtiLister
```
abstract class Lesson extends MtiListener {
```
2. Create a member with type of MtiListener
```
private static MtiListener mtiListener = new MtiListener();
```
3. Create the method for providing the listener
```
protected MtiListener getMtiListener() {
    return this.mtiListener;
}
```

With the last steps your app is enabled ‚receiving‘ callbacks raised by MTI and providing the informations to lessons. To get provided informations lessons have to register.

HowTo: Initialize the MTI Library

Your app is ready to receive callbacks. But MTI does not yet know where to send the callbacks to. This will be done with the next steps.

Again we use the class **Lesson1_Initialize** for implementation:

1. Create a member which we use later for checking the initialization status
```
private boolean statusInitialized = false;
```
2. Register the lesson for callbacks
```
public void Lesson1_Initialize(int functionId, String buttonCaption,
Fragment fragment) {
    super(functionId, buttonCaption, fragment);
    registerLesson(this);
}
```

3. Complete the method doSomething() calling registerListener()
   The method registerListener() will be implemented below.

   ```
   @Override
   public void doSomething() {
       registerListener();
       startMapTrip();
   }
   ```

   Please note that the new call registerListener() is the first line of doSomething(), startMapTrip() is the second one. The reason is that we want to be notified when MapTrip started. And notifying requires listening per callbacks.

4. Implement the method registerListener

   ```
   private void registerListener() {
       MTIHelper.initialize(fragment.getContext());
       Api.registerListener(getMtiListener());
   }
   ```

   Initializing the MTIHelper enables interacting MTI with your app.
   More interesting is the call Api.registerListener(). The method getMtiListener() delivers the class MtiListener which we created before. This is the point where we tell MTI which class is the callback class.

From now the app can ‚catch' callbacks from MTI.
But still we are missing some steps, especially the MTI initialization.

- In **Lesson1_Initialize.java** implement the API init

  ```
  private void initMTI() {
      if (statusInitialized) {
          Api.uninit();
          MTIHelper.uninitialize();
      }

      Api.init();
  }
  ```

But who or what calls this method?

In **Lesson1_Initialize.java** override some MtiListener callbacks.

1. Override the callback infoMsg()
   MapTrip sends many and different messages to callbacks. At this point the app waits for the messages that MapTrip was started or restored.

```
@Override
public void infoMsg(Info info, int i) {
    switch (info) {
        case MAPTRIP_STARTED:
        case MAPTRIP_WAS_RESTORED:
            initMTI();
            break;

        default:
            break;
    }
}
```

2. Override the callback initResult()
   The callback initResult() is called with successfull initialization of the API.

```
@Override
public void initResult(int requestId, ApiError apiError) {
    switch (apiError) {
        case OK:
            statusInitialized = true;
            break;

        default:
            // something to do if an error occured
            statusInitialized = false;
            break;
    }
}
```

Again you will rightly ask who or what is calling this methods.
With the last step we broadcast incoming MTI callbacks in the MtiListener to registered lessons.

1. Complete the callback infoMsg()

```
@Override
public void infoMsg(Info info, int i) {
    for (Lesson lesson : registeredLessons) {
        lesson.infoMsg(info, i);
    }
}
```

2. Complete the callback initResult()

```
@Override
public void initResult(int requestId, ApiError apiError) {
    for (Lesson lesson : registeredLessons) {
        lesson.initResult(requestId, apiError);
    }
}
```

That’s it. If you want to check if the initialization has any effect to MapTrip or the application, set a breakpoint to the new methods of Lesson1_Initialize.

After building and starting the app click again at the button START MAPTRIP AND INIT MTI and you will see that at the end the initResult() of Lesson1_Initialize will be called and – hopefully – statusInitialized is true.

Abstract

At the end of this lesson you get a simple app which starts MapTrip and initializes the MTI API. From now your app is able to communicate with and to control MapTrip.

You can find the described code fragments also in the folder of Lesson1_Initialize. This are:

- build.gradle (stores dependencies to the MTI lib and other libs)
- Lesson.java (basic class for lessons, delivers the MtiListener)
- Lesson1_Initialize.java (starting MapTrip and initialization of the MTI API)
- HomeFragment.java (method list with lessons)
- MtiListener.java (the callback class for MTI messages)

The complete github project is hosted here: https://github.com/MapTrip-Navigation/MTI-Tutorial