Lesson 1:

Setup Project, Include 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.

## Integration of 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](mailto: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 Projekt with your Android Studio**

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

## HowTo: Including MTI API Library

The MapTrip API is an Android Archive file (aar). 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.

1. Change to the *app* folder of the project
2. Create a new folder *libs*
3. Copy the file *mti.aar* from folder *Lessons/Lesson1\_Initialize/* to the new folder libs
4. In Android Studio open the file build.gradle (Module: app) and complete the dependencies:

dependencies {

implementation fileTree(**dir**: **"libs"**, **include**: [**"\*.jar"**, **"\*.aar"**])

}

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

Now the MTI lib (mit.aar) is physically part of your project and can be considered while project compilation.

## 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**Also implement a constructor for later usage:  
     
   **public class** Lesson1\_Initialize **extends** Lesson {  
     
    **public** Lesson1\_Initialize(**int** functionId, String buttonCaption, Fragment fragment) {  
    **super**(functionId, buttonCaption, fragment);  
    }  
    …  
   }
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*  
   Lesson lesson = new Lesson1\_Initialize(lessonIndex++,"Start MapTrip And Init MTI", this);  
    lessonArrayList.add(lesson);**  
     
    **return** lessonArrayList;  
   }
4. Override the method doSomething() of base class Lesson  
   This method will be called when the button START MAPTRIP AND INIT MTI is clicked.  
   @Override  
    **public void** doSomething() {  
    startMapTrip();  
    }
5. Implement the method startMapTrip()  
   This method creates an Intent and starts the Activity   
   **private boolean** startMapTrip() {  
    Intent intent = **fragment**.getActivity().getPackageManager().getLaunchIntentForPackage(**"de.infoware.maptrip.navi.license"**);  
    **if** (**null** == intent) {  
    **return false**;  
    }  
     
    **try** {  
    **fragment**.getActivity().startActivity(intent);  
    } **catch** (ActivityNotFoundException eToo) {  
    *//*  
   } **catch** (Exception ex) {  
    *//*  
   }  
     
    **return true**;  
   }

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

## HowTo: Initialize the MTI Library

Last step of our first lesson is initializing the MTI lib. With the initialization we tell the lib which Context of your App receives the MTI callbacks. You remember, the lib uses callbacks to send responses to function calls or informations about MapTrip status.

The MTI callback interface for basic functions is the ApiListener.

1. Make the class Lesson as an implementation of ApiListener by extending the class with the implements keyword:  
     
   **public abstract class** Lesson **implements ApiListener** {...  
     
   In the next moment asks AndroidStudio asks if it should create the methods of the interface. After that you will see the interface methods at the end of the class.   
     
   For later use also add the following member to the class:  
     
   **protected static boolean** *statusInitialized* = **false**;
2. Create a new function for init in the class **Lesson1\_Initialize.java**  
     
   **private void** initMTI() {  
     
    **if** (*statusInitialized*) {  
    Api.*uninit*();  
    MTIHelper.*uninitialize*();  
    }  
     
    MTIHelper.*initialize*(**fragment**.getContext());  
    Api.*registerListener*(**this**);  
     
    **int** requestId = Api.*init*();  
   }
3. There is no point in initializing the API multiple times. Before we call a new init it is recommended to make an uninit before. To control this we used the boolean variable **statusInitialized**. But where it is switched to TRUE?   
   MTI sents the response to the call Api.init() to the callback method initResult(). There we can check the success and set the **statusInitialized**:  
     
   @Override  
   **public void** initResult(**int** requestId, ApiError apiError) {  
    **if** (apiError.equals(ApiError.***OK***)) {  
    *statusInitialized* = **true**;  
    }  
   }

That’s it. If you want to check if the initialization has any effect to MapTrip or the application, set a breakpoint to the callback method Lesson.initResult().

After building and starting the app click again at the button START MAPTRIP AND INIT MTI and you will see that MapTrip reacts to the init call.

## 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 (implementation of MTI Callbacks and some members)
* Lesson1\_Initialize.java (starting MapTrip and initialization of the MTI API)
* HomeFragment.java (method list with lessons)

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