Gurtam created WiaTagKit library that allows mobile devices to communicate with Wialon platform. It means that now you can easily add tracker functionality to any of your mobile apps or just develop a new one.

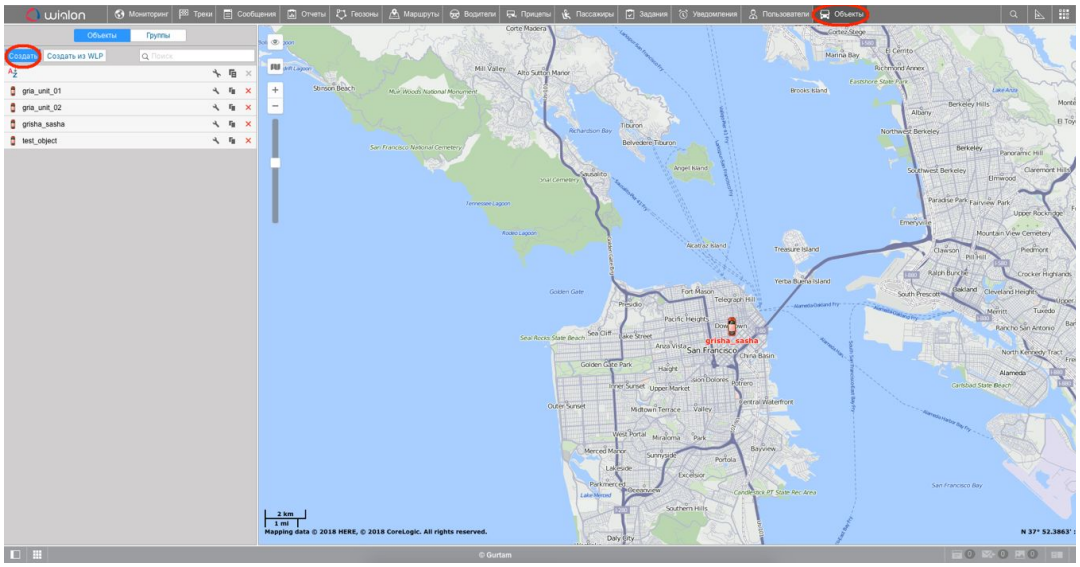
Why should you use the new WiaTagKit library?

- **New closed protocol**. It is more secure, up-to-date, and compact (if compared to the old GPSTag). It consumes less traffic and is harder to hack.

- **You get a ready-made component** which means you reduce the cost and the time of the development. Thus, your solutions appear on the market faster and you don't have to put efforts into Wialon connection protocol development.

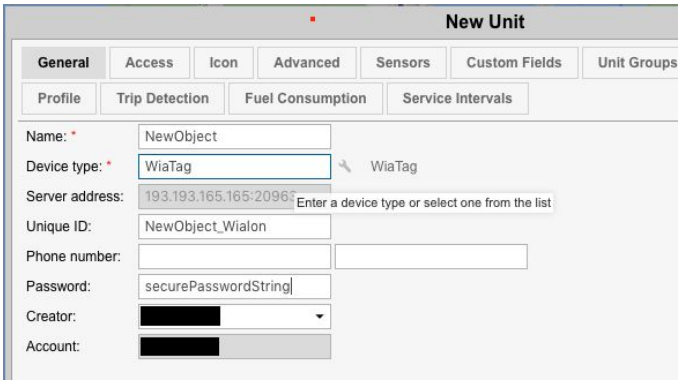
Android Manual

1. Create the unit that will send messages to Wialon: access your user account, open the **"Units"** tab, press the **"New"** button.



2. Set up the unit in the popped-up dialog: specify the **Name**, select the WiaTag **Device type**, fill in the **Unique ID** and **Password** fields.

Note! Anyone with the unit's ID and password can send messages to Wialon on behalf of this unit. Make sure that only authorized people know the password.



To send the message, you will need the **Server address**, **Unique ID** and **Password** fields.

Lets have some coding

Follow these steps to create a new app project including an empty activity:

1. Start **Android Studio**.
2. Create a new project in the **“Welcome to Android Studio”** dialog or select File -> New -> New Project on the menu bar.
3. Enter your app name, company domain, and the project location as prompted. Then click Next.
4. Select the form factors you need for your app. If you are not quite sure what form factors you need, just select **Phone** and **Tablet**. Then click Next.
5. Select **”Empty Activity”** in the **“Add an activity to Mobile”** dialog. Then click Next.
6. Enter the **Activity** name, layout name and title as prompted. The default values are fine. Then click Finish.

To make the **wiatag-kit library** available to your app:

1. Open the build.gradle file inside your application module directory

Note! Android Studio projects contain a top-level **build.gradle** file and a **build.gradle** file for each module. Make sure you edit the file for your application module.

2. Add a new build rule under dependencies for the latest version of wiatag-kit:

```
apply plugin: 'com.android.application'
...

dependencies {
    implementation "com.gurtam:wiatag-kit:0.1.4"
}
```

Note! Make sure your top-level build.gradle contains a reference to the jcenter() repo.

3. Save the changes, and click Sync Project with Gradle Files in the toolbar.

The next step is to add the correct Permission in **Android Manifest**. In order to perform network operations in your application, your manifest must include the following permissions:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Sending message to Wialon

1. Initialize **MessageSender** using the Wialon unit's **Server address**, **Unique ID**, and **Password**;
2. Initialize the Message unit;
3. Send **Message** using the corresponding **MessageSender** method;
4. Change the **MainActivity** class (if the default name is used).

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

import com.gurtam.wiatagkit.Message;
import com.gurtam.wiatagkit.MessageSender;
import com.gurtam.wiatagkit.MessageSenderListener;

import java.util.Date;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String host = "193.193.165.165";
        int port = 20963;
        String unitId = " NewObject_Wialon";
        String password = "securePasswordString";
        MessageSender.initWithHost(host,port,unitId,password);
        Message message = new Message().time(new Date().getTime());
        MessageSender.sendMessage(message,new MessageSenderListener() {
            @Override
            protected void onSuccess() {
                super.onSuccess();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(MainActivity.this,"Message
Sent",Toast.LENGTH_LONG).show();
                    }
                });
            }
            @Override
```

```
            protected void onFailure(byte errorCode) {
                super.onFailure(errorCode);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {

Toast.makeText(MainActivity.this,"Failure",Toast.LENGTH_LONG).show();
                    }
                });
            }
        });
    }
}
```

Note! The **host**, **port**, **unitId** and **password** values should be replaced by your WiaTag unit's **Server Address (IP, port)**, **Unique ID,** and **Password**.

Launch the project.

If you did everything right, you will see the message: **Toast Message Sent**
It means you sent the first empty message to the server.

Adding the content to the message to WiaTag

Currently, the message can contain the following information:

1. The time of the message creation measured in milliseconds since 1 January 1970. Required parameter.
2. Location. The *Location* class is used to send the location.
3. LBS data. The *Lbs* class is used to send the LBS data.
4. SOS-message flag.
5. Image.
6. Text message.
7. Battery charge level.
8. Parameters. They have the keys (only text ones) and the values (text, binary parameters and the ones of the **Int**, **Long**, **Float**, **Double** types). You can't send 2 parameters for one and the same value of the key.

For example, let's create the message that sends time, SOS-signal, text message, and Int-parameter:

```
Message message = new Message()
.time(new Date().getTime())
.Sos()
.location(new Location(53.90582,27.45697,290,2F,(short)15,(byte)8));
.text("This is my text message!")
.addParam("int value", 3);
```

You will get callback through *MessageSenderListener* passed to the *sendMessage* method of the *MessageSender* class.

If successful *onSucces* method will be called, otherwise – *OnFailure* with the error code:

```
FAILED_TO_CONNECT = 3;
FAILED_TO_SEND = 4;
INVALID_UNIQUE_ID = 5;
INCORRECT_PASSWORD = 6;
INCORRECT_MESSAGE = 8;
```

The example of sending one or several messages is available at GitHub.

It is one of the examples of how to work with **WiaTagKit.** Feel free to ask any questions on the new library usage contacting us at development@gurtam.com.