

Использование контроллера ФН (ТМ) с Android аппаратами



5. авг. 2025

<mailto:nilstarsoft@mail.ru>

www.nilsoft.ru

Оглавление

1 Введение.....	3
2 Общая схема Android аппарата с контроллером ФН.....	4
3 Работа с android сервисом, встроенным в программу «ККТ сервис».....	6
3.1 Запуск сервиса.....	6
3.2 Связывание приложения с сервисом.....	7
3.3 Обмен данными с сервисом.....	8
3.3.1 Обращение к сервису.....	9
3.3.1.1 Пример отправки сообщения с командой к сервису.....	9
3.3.1.2 Пример отправки сообщения с текстом для печати к сервису.....	10
3.3.2 Получение ответов от сервиса.....	10
4 SDK для работы с контроллером ФН.....	13
4.1 Комплект поставки.....	13
4.2 Требования.....	13
4.3 Использование библиотеки.....	14
4.3.1 Создание приложения.....	14
5 Краткое описание интерфейсных классов библиотеки.....	19
5.1 Обращение к интерфейсным классам модуля.....	19
5.2 Использование TMLib.....	20
5.3 Использование TMLib.....	21
5.3.1 Получение текстовой интерпретации ошибок контроллера ФН.....	21
5.3.2 Получение текстовой интерпретации ошибок системы печати.....	21
5.4 Использование TMLib.....	21
5.4.1 Инициализация класса.....	22
5.4.2 Работа с контроллером ФН из активностей.....	22
5.5 Использование TMLibHandler.....	23
5.5.1 Использование класса TMLibKeysUpdate (ФФД 1.2 с TMT).....	24
5.5.2 Использование класса TMLibOismHandler (ФФД 1.2 с TMT).....	25
5.6 Использование потока фоновой отправки документов в ОФД.....	25
5.7 Использование потока фоновой отправки документов в ОИСМ (ФФД 1.2 с TMT).....	25

1 Введение

Данный документ описывает работу контроллера ФН (в том числе и программного контроллера) с Android аппаратами. Работа с программным и аппаратным контроллером фактически не отличается (за несколькими нюансами, о которых в документе будет отдельно замечено), поэтому, предлагаемое SDK никак не отличается, при использовании на разных типах контроллеров и Android аппаратах.

В этом документе не рассматривается система команд контроллера ФН (система команд контроллера описана в документе «Руководство по программированию контроллера ФН» - доступен у производителя контроллера ФН).

Внимание! Реализация контроллера ФН в виде отдельного законченного изделия позволяет решать следующие задачи:

- Не использовать ресурсы Android аппарата при работе с ФН (т. е. контроллер ФН обеспечивает работу по протоколу с ФН, не используя процессорное время и память Android аппарата);
- Блокирование работы Android в результате внутреннего сбоя или работы ресурсоемкого приложения не влияет на работу контроллера ФН;
- Не реагировать на изменение времени при синхронизации Android аппарата по внешним сетевым устройствам, так как контроллер ФН использует собственный часы/календарь.

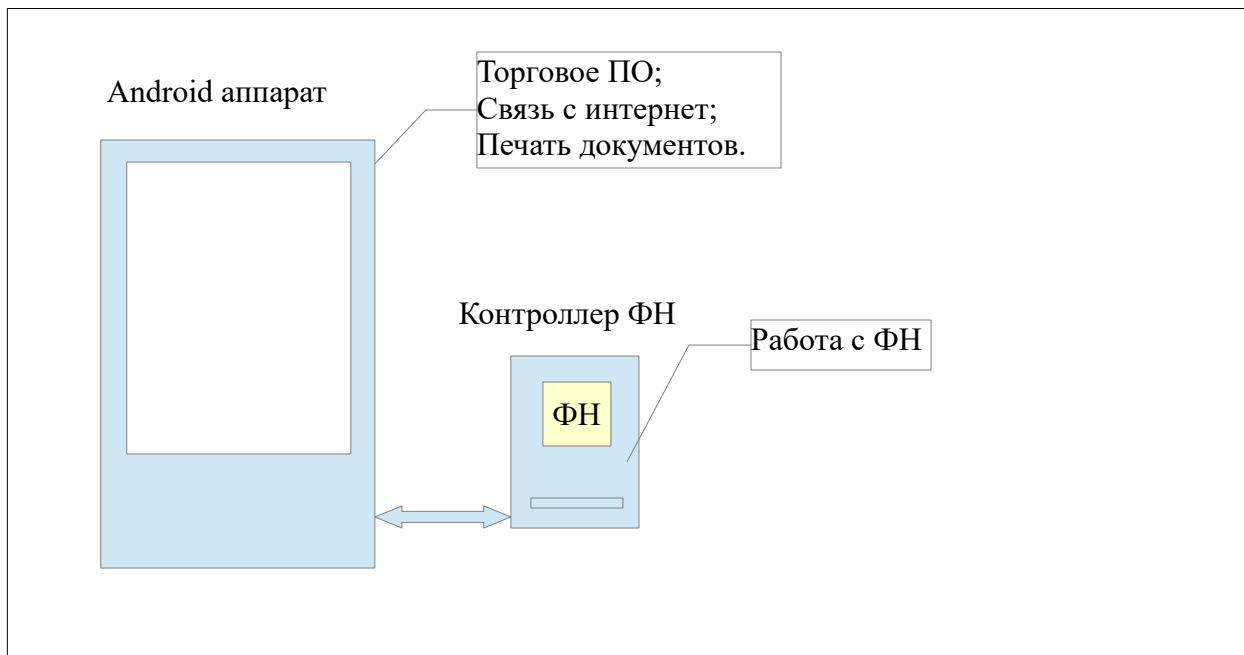
Для обслуживания контроллера ФН производителем поставляется сервисная программ ККТ, позволяющая производить все операции над контроллером ФН (программа ККТ является либо предустановленной программой либо доступна в составе магазина приложений).

Внимание! Программа «ККТ сервис» (на Andoird: «ККТ») включает в себя сервис, предоставляющий интерфейс для работы с контроллером ФН (или с программным контроллером). Внешние программы работающие с предоставляемым SDK используют этот сервис для обращения к контроллеру ФН.

2 Общая схема Android аппарата с контроллером ФН

Ниже приведена схема соединения Android аппарата с контроллером.

Внимание! В случае использования программного контроллера ФН, программная часть находится в Android аппарате, как часть приложения «ККТ сервис».



Внимание! Контроллер ФН не оснащен собственной сетевой платой (УПД) и печатающим устройством, поэтому для печати чеков и отправки контейнеров в ОФД используется соответствующее оборудование Android аппарата. Программный контроллер ФН, может иметь реализацию функционала УПД — уточняйте по документации на конкретную реализацию.

Контроллер ФН решает следующие задачи:

- Работу с ФН (содержит разъем для установки ФН);
- Прием команд по протоколу и подготовка ответов;
- Подготовка печатных форм и передача их торговому ПО;
- Подготовка контейнеров для отправки в ОФД;
- Прием и обработка ответов из ОФД;
- Хранение настроек контроллера ФН (настройки ОФД, параметры документов, вид заголовка и завершения документа).

Внимание! Контроллер ФН, для повышения надежности работы, оснащен своими часами/календарем (RTC). При начале работы необходимо синхронизировать время (необходимо учитывать правила работы с кассовым оборудованием). При пересечении различных часовых поясов рекомендуется не использовать время Android аппарата (которое может синхронизироваться автоматически по времени сети), а использовать внутренние часы контроллера ФН.

Внимание! Программный контроллер ФН использует часы Android аппарата — поэтому надо отнестись с большей внимательностью при установке времени на Android аппаратах.

Торговое ПО, установленное на Android аппарате должно решать следующие задачи:

- Подготовка команд и обработка ответов от контроллера ФН;
- Печать печатных форм, полученных от контроллера ФН, на принтере;
- Отправка контейнеров документов из ФН в ОФД и загрузка квитанций в контроллер ФН.
- Обновление ключей на АС ОКП (для ФФД 1.2 с установленной поддержкой ТМТ);
- Отправка контейнеров документов из ФН в ОИСМ и загрузка квитанций в контроллер ФН (для ФФД 1.2 с установленной поддержкой ТМТ).

Способ интеграции поддержки контроллера ФН в торговое ПО выбирает производитель ПО.

Для облегчения интеграции с торговым ПО в сервисную программу ККТ (программа для обслуживания контроллера ФН) включен сервис, предоставляющий интерфейс для работы с контроллером. Производителем поставляется SDK для работы с этим сервисом.

Внимание! В принципе, можно работать и напрямую с android сервисом (является частью программы «ККТ сервис»), без использования SDK. Но в этом случае нужно отслеживать все изменения законодательства и соответственно менять все свои структуры (согласно инструкции «Инструкция по программированию контроллера ФН и стационарных ККТ¹»). При использовании SDK, все обновления выходят в большинстве случаев с новой версией программы «ККТ сервис» и доступны на сайте разработчика ККТ.

1 Инструкция доступна на сайте www.nilsoft.ru

3 Работа с android сервисом, встроенным в программу «ККТ сервис»

В этом разделе описана прямая работа с android сервисом, встроенным в программу «ККТ сервис».

Внимание! Производитель рекомендует использовать SDK для взаимодействия ККТ (смотри раздел 4).

Название android сервиса: **ru.nilsoft.www.kkt.TMService**

3.1 Запуск сервиса

Процедура запуска сервиса должна включать проверку, что сервис не запущен и запуск его если, он не запущен.

Пример процедуры запуска сервиса:

```
/** Название класса сервиса для работы с контроллером ФН. */
public final static String ServiceName = "TMService";
/** Название package к которому относится сервис для работы с контроллером ФН. */
public final static String ServicePkg = "ru.nilsoft.www.kkt";
/** Полное название класса сервиса для работы с контроллером ФН. */
public final static String ServiceFullName = ServicePkg+"."+ServiceName;

/**
 * Проверка, что сервис запущен.
 *
 * @return true: сервис найден, false: сервис не найден.
 * @param context контекст приложения
 */
private boolean isServiceRunning(Context context) {
    ActivityManager manager = (ActivityManager)context.getSystemService(Context.ACTIVITY_SERVICE);
    if ( manager != null ) {
        for (ActivityManager.RunningServiceInfo service : manager.getRunningServices(Integer.MAX_VALUE)) {
            if (ServiceFullName.equals(service.service.getClassName())) {
                return true;
            }
        }
    }
    return false;
}

/**
 * Запуск сервиса.
 *
 * @param context контекст приложения.
 * @return интент сервиса.
 */
public Intent startService(Context context) {
    //запуск сервиса
    ComponentName component = new ComponentName(ServicePkg, ServiceFullName);
    Intent service = new Intent(ServiceFullName);
    service.setComponent(component);

    boolean isRun = isServiceRunning(context);

    //запускаем сервис только если он не был запущен ранее
    if ( !isRun ) {
        try {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                //Android Version >= Oreo (8.0) has limited background activities.
                context.startForegroundService(service);
            }
        }
    }
}
```

```

        } else {
            context.startService(service);
        }
    }
    catch(Exception ex) {
        ex.printStackTrace();
    }
}

return service;
}

```

Работа с сервисом производится путем обмена сообщениями.

Внимание! Для отправки сообщений к сервису нужно добавить в manifest.xml следующие строки (они дают доступ к сервису в системе Android):

```

<queries>
  <package android:name="ru.nilsoft.www.kkt" />
</queries>

```

3.2 Связывание приложения с сервисом

Для работы с сервисом, необходимо осуществить связывание приложения с сервисом.

Пример процедуры привязывания приложения с сервисом:

```

/** Привязка к сервису. */
private Messenger mBind = null;
/** Класс для получения сообщений от сервиса. */
private MyReceiver myReceiver = null;

/** Класс для получения связки с сервисом. */
private final ServiceConnection mConnection = new ServiceConnection() {
    /**
     * Процедура срабатывающая при привязке к сервису.
     *
     * @param className имя сервиса.
     * @param service биндер сервиса.
     */
    public void onServiceConnected(ComponentName className, IBinder service) {
        // This is called when the connection with the service has been
        // established, giving us the object we can use to
        // interact with the service. We are communicating with the
        // service using a Messenger, so here we get a client-side
        // representation of that from the raw IBinder object.
        mBind = new Messenger(service);

        // здесь обработка успешного соединения с сервисом
        ...
    }
}

/**
 * Процедура срабатывающая при потере связи с сервисом.
 *
 * @param className имя компонента.
 */
public void onServiceDisconnected(ComponentName className) {
    // This is called when the connection with the service has been
    // unexpectedly disconnected -- that is, its process crashed.
    mBind = null;
}
};

/**
 * Привязка к сервису.

```

```

*
* @return true: привязка к серверу произошла, false: привязки к сервер нет.
* @param context контекст приложения.
*/
public boolean bind(Context context) {
    Intent service = startService(context);

    //регистрация класса BroadcastReceiver для получения сообщений от сервиса
    IntentFilter intentFilter = new IntentFilter();
    intentFilter.addAction(ServiceName); //no имени класса сервиса
    intentFilter.addCategory(Intent.CATEGORY_DEFAULT);

    myReceiver = new MyReceiver();

    if( Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        //регистрируем ресивер для получения сообщений от сервиса, расположенного в другом приложении
        context.registerReceiver(myReceiver, intentFilter, Context.RECEIVER_EXPORTED);
    }
    else {
        context.registerReceiver(myReceiver, intentFilter);
    }

    //привязка к сервису
    return context.bindService(service, mConnection, 0);
}

```

Где класс **MyReceiver** - это класс, наследуемый от **BroadcastReceiver** для приема ответов от сервиса ККТ.

```

/** Класс для получения сообщений от сервиса. */
private class MyReceiver extends BroadcastReceiver {
    /**
     * Процедура срабатывающая при получении широковещательного сообщения от сервиса.
     *
     * @param arg0 контекст.
     * @param arg1 интент.
     */
    @Override
    public void onReceive(Context arg0, Intent arg1) {
        Message msgTM = arg1.getParcelableExtra("resp");

        if (msgTM == null) return;
        if (BuildConfig.DEBUG)
            Log.d(this.getClass().getCanonicalName(), "Receive what:" + msgTM.what + " arg1:" + msgTM.arg1 + " arg2:" +
            msgTM.arg2);

        //здесь обработка, полученных данных от сервиса
        ...
    }
}

```

3.3 Обмен данными с сервисом

ККТ сервис поддерживает следующие типы запросов:

- Прием команд к контроллеру ФН;
- Прямое обращение к контроллеру ФН, без предварительной обработки команд. Например это нужно для работы с контроллером ФН в режиме сервисного монитора (использовать этот режим работы сторонним приложениям нет надобности);
- Работа с системой печати (если аппарат поддерживает печать).

Внимание! Ответы от «ККТ сервиса» приходят не сразу, а могут занимать некоторое время, поэтому если Ваша функционал построен по принципу обращение-ответ внутри одной

функции, то нужно запускать функцию в отдельном потоке. Также нужно учитывать, что ответ не обязателен от сервиса, и поэтому необходимо заложить таймаут ожидания при работе с сервисом.

3.3.1 Обращение к сервису

Константы, используемые при обращении к сервису ККТ, определены в файле **TMDefaults.java** (смотрите описание SDK), здесь приведены только основные, необходимые в работе константы.

```
/**
 * <H1>Класс определений для сообщений, используемых для обмена с сервисом ККТ.</H1>
 */
public final class TMDefaults {
    /** Типы сообщений. */
    public final static class msg_type {
        /** Старт сервиса (используется только для Android 6.0 и выше). */
        public final static int START = 0;
        /** Запрос к сервису. */
        public final static int REQ = 1;
        /** Ответ сервиса. */
        public final static int RESP = 2;
        /** Запрос сквозного режима. */
        public final static int REQ_BYPASS = 3;
        /** Ответ сквозного режима. */
        public final static int RESP_BYPASS = 4;
        /** Запрос на печать. */
        public final static int REQ_PRN = 5;
        /** Ответ модуля печати. */
        public final static int RESP_PRN = 6;
        /** Передать данные для записи в лог. */
        public final static int REQ_ADD2LOG = 7;
    }

    /** Подтипы сообщений (для {@link msg_type#REQ}/{@link msg_type#RESP}). */
    public final static class msg_subtype{
        /** Инициализация КФН. */
        public final static int INIT = 0;
        /** Отправка/ответ команды КФН. */
        public final static int TM = 1;
        /** Отправка/ответ произвольных данных КФН. */
        public final static int DATA = 2;
        /** Информация о сервисе. */
        public final static int INFO = 3;
        /** Версия сервиса. */
        public final static int VER = 4;
        /** Ошибка сервиса. */
        public final static int ERR = 5;
    }

    // прочие константы
    ...
}
```

3.3.1.1 Пример отправки сообщения с командой к сервису

```
//отправка данных в сервис ФН
Bundle bundle = new Bundle();
bundle.putByteArray("data", data);
Message msg = Message.obtain(null, TMDefaults.msg_type.REQ, TMDefaults.msg_subtype.TM, 0);
msg.setData(bundle);

try {
    //отправка команды
```

```

    if (BuildConfig.DEBUG) Log.d(this.getClass().getSimpleName()+" "+this, "mBind.send: " + cmd);
    mBind.send(msg);
}
catch (Exception e) {
    //ошибка отправка команды
    e.printStackTrace();
}

```

Где **data** – байтовый массив данных с командой обращения к контроллеру ФН.

Внимание! Описание команд доступно в инструкции «Инструкция по программированию контроллера ФН и стационарных ККТ²»

3.3.1.2 Пример отправки сообщения с текстом для печати к сервису

```

// отправка данных в сервис ФН
Bundle bundle = new Bundle();
bundle.putByteArray("data", data);
Message msg = Message.obtain(null, TMDDefaults.msg_type.REQ_PRN, TMDDefaults.msg_prntype.DATA, 0);
msg.setData(bundle);

try {
    //отправка команды
    mBind.send(msg);
}
catch (Exception e) {
    //ошибка отправки данных для печати
    e.printStackTrace();
}

```

Где **data** – данные для печати.

Внимание! Система печати эмулирует EPSON-совместимый принтер с некоторыми ограничениями (т. е. не все управляющие последовательности поддерживаются).

3.3.2 Получение ответов от сервиса

Для получения ответов от сервиса определяется класс `MyReceiver`, который используется при связывании приложения с сервисом.

Пример класса с обработкой ответов от сервиса:

```

/** Класс для получения сообщений от сервиса. */
private class MyReceiver extends BroadcastReceiver {
    /**
     * Процедура срабатывающая при получении широковещательного сообщения от сервиса.
     *
     * @param arg0 контекст.
     * @param arg1 интент.
     */
    @Override
    public void onReceive(Context arg0, Intent arg1) {
        Message msgTM = arg1.getParcelableExtra("resp");

        if (msgTM == null) return;
        if (BuildConfig.DEBUG) Log.d(this.getClass().getCanonicalName(), "Receive what:" + msgTM.what + " arg1:" + msgTM.arg1
+ " arg2:" + msgTM.arg2);

        switch(msgTM.what) {
            case TMDDefaults.msg_type.RESP:
                //ответ от КФН
                switch(msgTM.arg1) {
                    case TMDDefaults.msg_subtype.INIT:

```

2 Инструкция доступна на сайте www.nilsoft.ru

```

        //КФН успешно инициализирован

        // обработка данных приложением
        ...
        break;

    case TMDDefaults.msg_subtype.TM:{
        //ответ на команду (или на NAK)
        Bundle b = msgTM.getData();
        //данные ответа на команду
        byte[] resp = b.getBytes("data");

        // обработка данных приложением
        ...
    }
    break;

    case TMDDefaults.msg_subtype.DATA:{
        //ответ на DLE
        Bundle b = msgTM.getData();
        byte[] resp = b.getBytes("data");

        // обработка данных приложением
        ...
    }
    break;

    case TMDDefaults.msg_subtype.ERR:{
        //ошибка обращения к КФН
        Bundle b = msgTM.getData();

        //сообщение об ошибке
        String errorMsg = b.getString("error");

        // обработка данных приложением
        ...
    }
    break;

    case TMDDefaults.msg_subtype.VER:{
        //ответ на запрос версии

        //получен ответ о версии сервиса КФН
        Bundle data = msgTM.getData();
        if (data != null) {
            int hi_ver = data.getInt("hi_ver", 0);
            int mi_ver = data.getInt("mi_ver", 0);
            int lo_ver = data.getInt("lo_ver", 0);
            int date_ver = data.getInt("svn_ver", 0);
            String t = data.getString("version", "");

            // обработка данных приложением
            ...
        }
    }
    break;

    case TMDDefaults.msg_type.RESP_BYPASS:{
        //ответ в прямом режиме работы с КФН
        Bundle b = msgTM.getData();
        byte[] data = b.getBytes("data");
        if ( ( data != null ) && (data.length > 0 ) ) {
            // обработка данных приложением
            ...
        }
    }
    break;

```

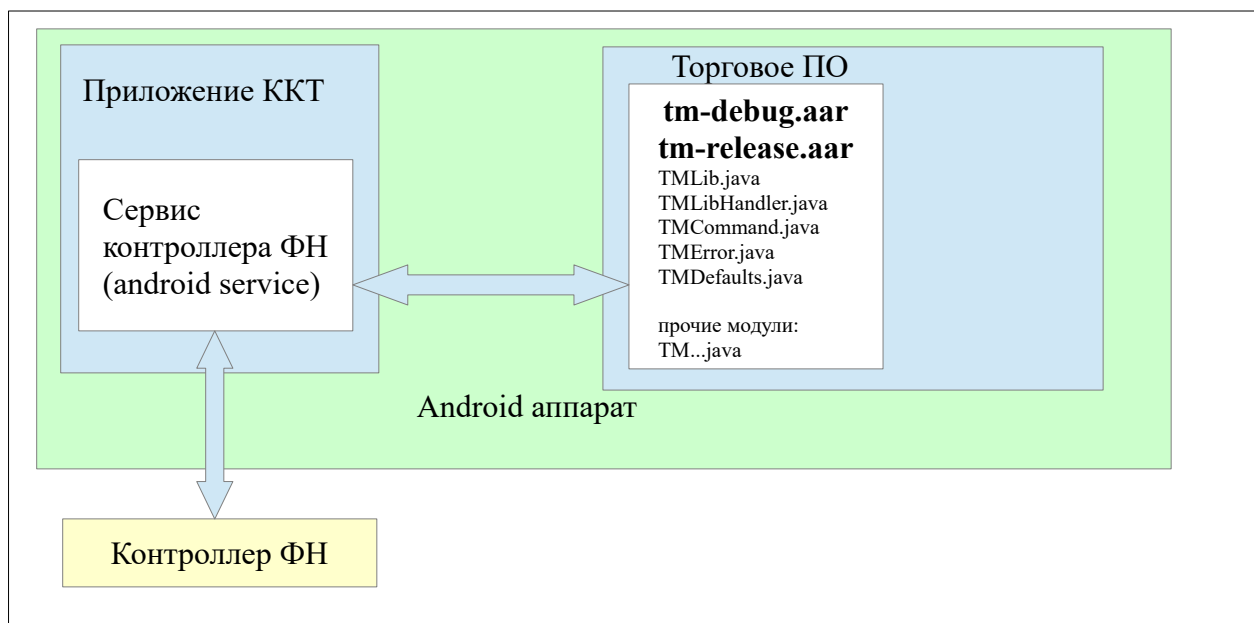
```
case TMDDefaults.msg_type.RESP_PRN:{  
    //ответ от принтера  
    int main_err = msgTM.arg1;  
    int sub_err = msgTM.arg2;  
  
    // обработка данных приложением  
    ...  
}  
break;  
}  
}
```

4 SDK для работы с контроллером ФН

Производитель предоставляет SDK высокого уровня для работы с контроллером ФН. SDK является универсальным и не зависит от используемого аппарата).

SDK основано на использовании сервиса Android, работающего с контроллером ФН (сервис встроен в приложение ККТ, и это приложение обязательно должно быть установлено на аппарате).

Схема использования SDK:



4.1 Комплект поставки

SDK для разработчика поставляется в следующей комплектации:

1. **tm-debug.aar** – android библиотека с интерфейсами для работы с сервисом (отладочная поставка), работающим с контроллером ФН (далее сервиса контроллера ФН);
2. **tm-release.aar** – android библиотека с интерфейсами для работы с сервисом (релизная поставка), работающим с контроллером ФН (далее сервиса контроллера ФН);
3. **tm.doc.zip** – документация по интерфейсным классам api сервиса контроллера ФН (сгенерирована JavaDoc);
4. **example.zip** – исходные коды образцового приложения.

Внимание! Библиотеки **tm-debug.aar** и **tm-release.aar** рассчитана на минимальные требования к версии SDK Android: **API 21: Android 5.0**.

4.2 Требования³

Библиотека **tm-debug** для системы android поставляется в виде **aar**-библиотеки для Android Studio.

Рекомендуется к использованию средство разработки **Android Studio** версии не ниже 3.0.

³ Сверяйтесь с текущей версией readme.txt из состава SDK – так как описание может отставать от реализации.



Внимание! Для разработки и отладки приложений аппарат должен отвечать следующим условиям:

- Должна быть включен режим отладки на Android аппарате;
- Android аппарат должен находиться в «**DEBUG only**» режиме, чтобы была возможность запускать ПО в отладочном режиме на аппарате. Когда аппарат находится в этом режиме, то на экране отображается значок отладочного режима (зависит от производителя Android аппарата).

4.3 Использование библиотеки

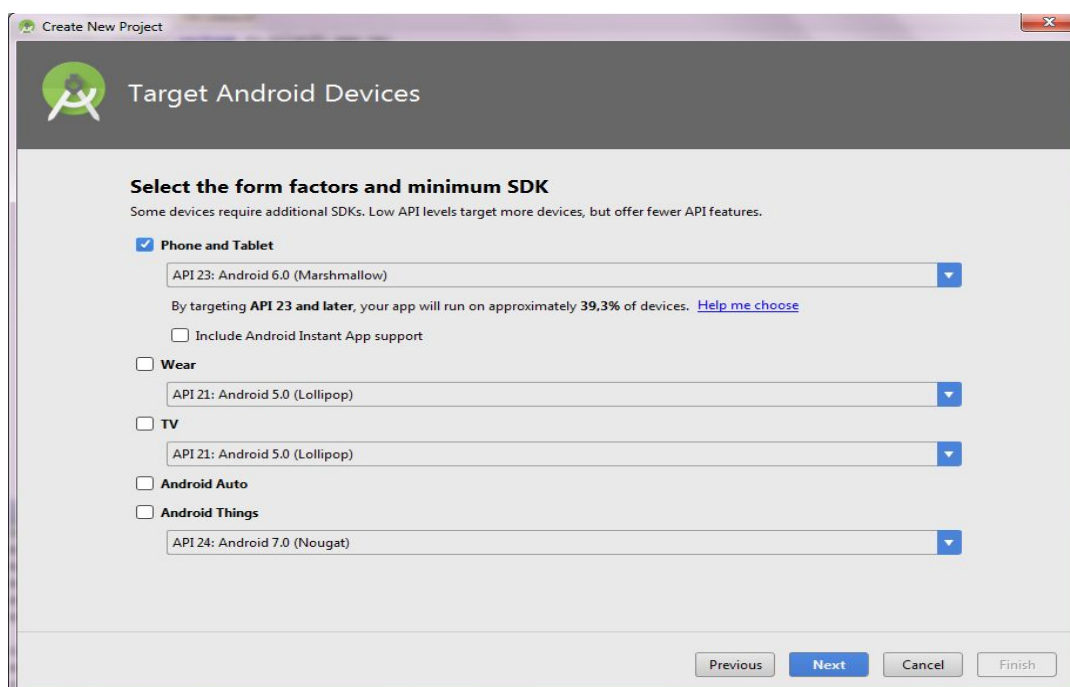
Далее описывается использование библиотеки.

Сам модуль работы с контроллером ФН является сервисом и работает независимо от приложения. Обращение к сервису контроллера ФН происходит через библиотеку.

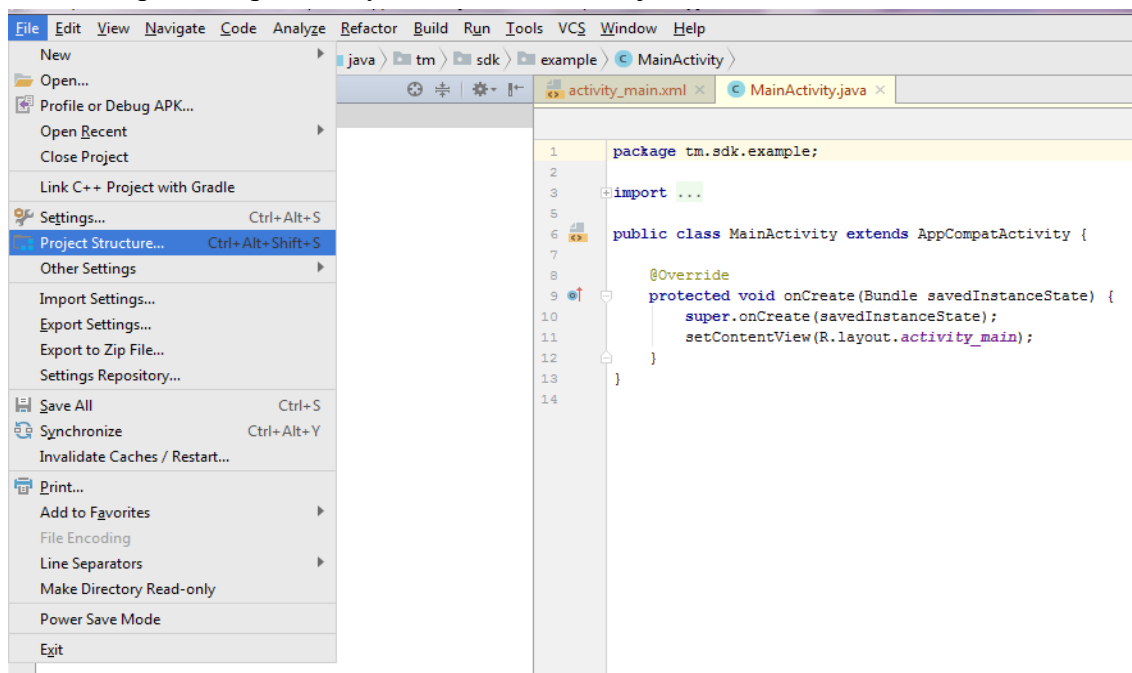
Подразумевается, что для разработки используется средство разработки **Android Studio**.

4.3.1 Создание приложения

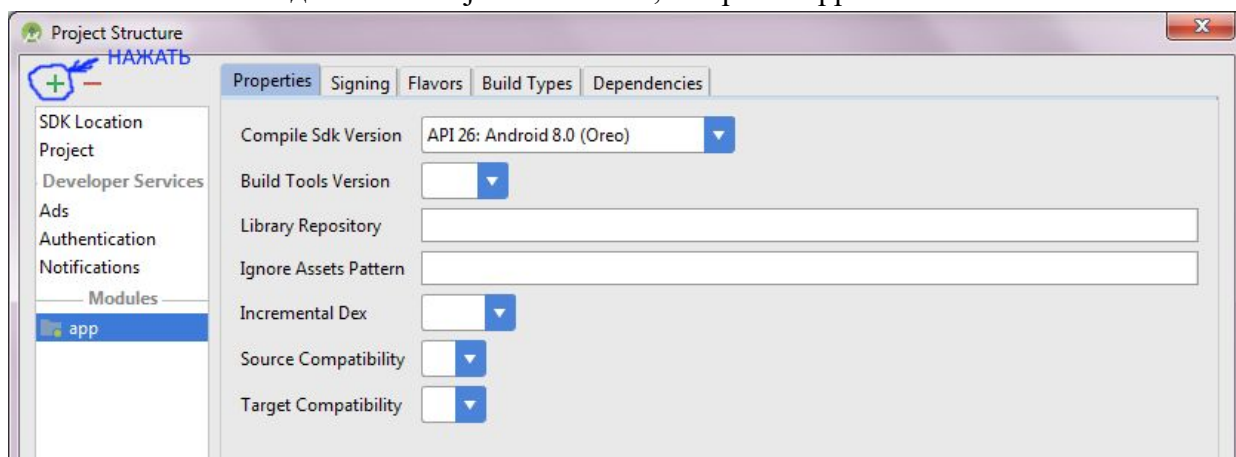
Создание приложения с использованием библиотеки **tm-debug.aar** (используется **Android Studio 3.01**):



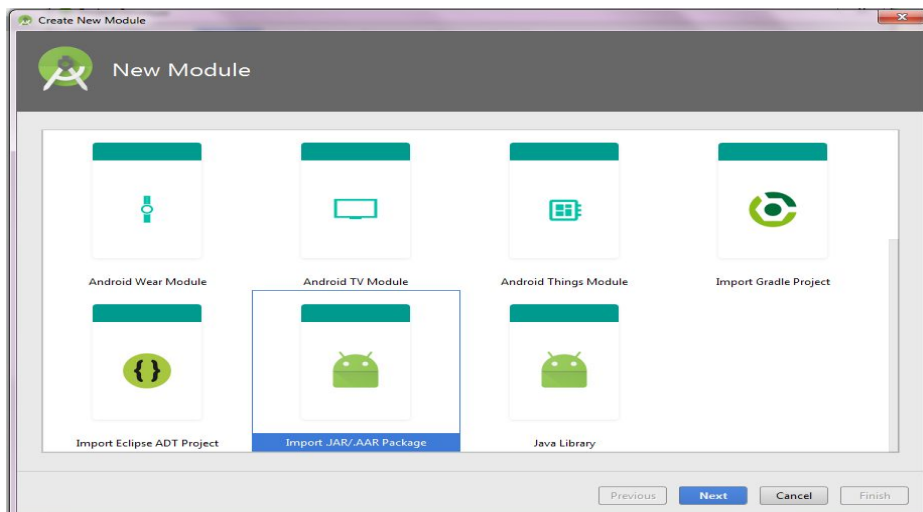
1. Создаем проект, при создании проекта нужно учитывать, что будем использовать Android API не выше 6.0:
2. Выбираем в проекте пункт меню «File/Project Structure»:



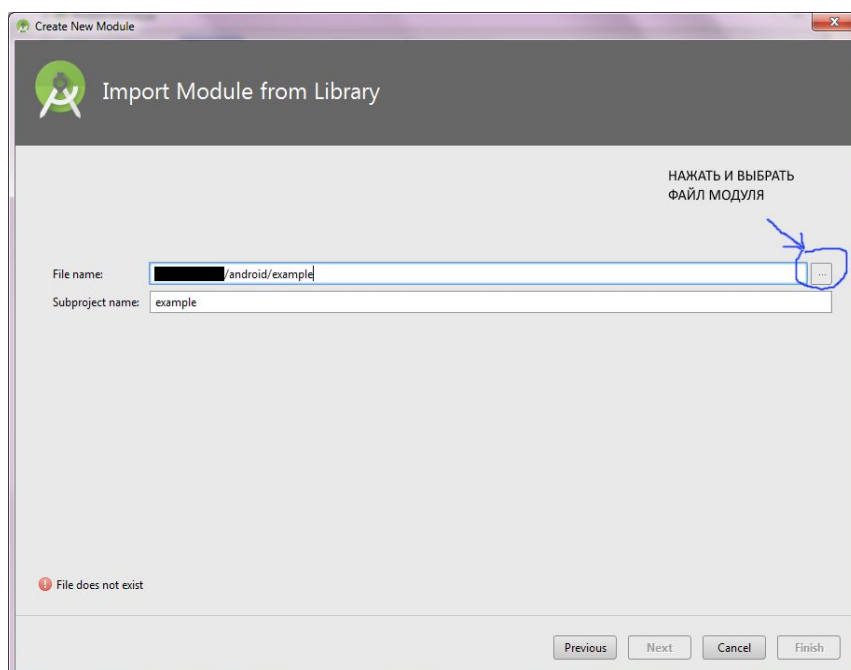
3. В появившемся диалоге «Project Structure», выбрать «app» и нажать «+»:



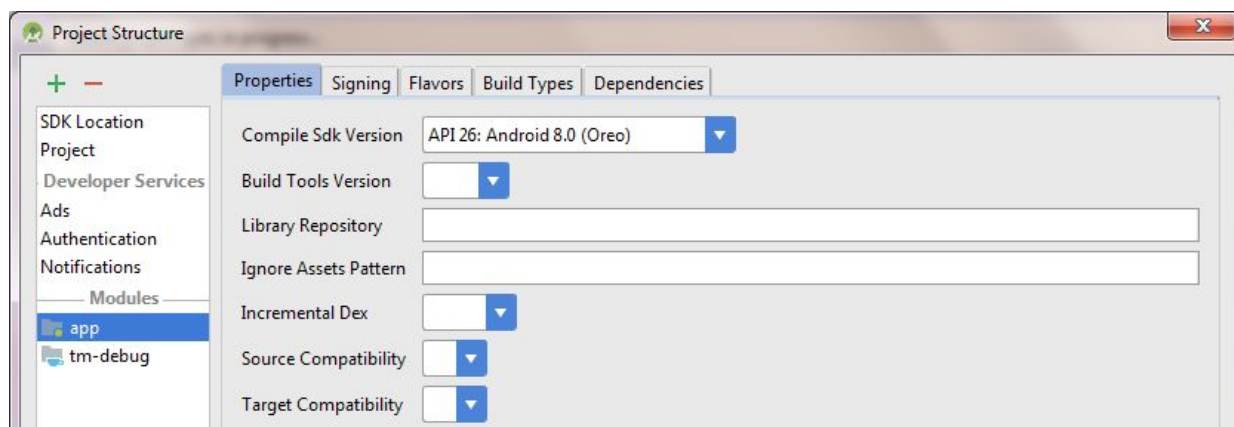
4. Появится диалог «Create New Module», в нем выбрать шаблон «Import JAR/AAR package» и нажать кнопку «Next»:



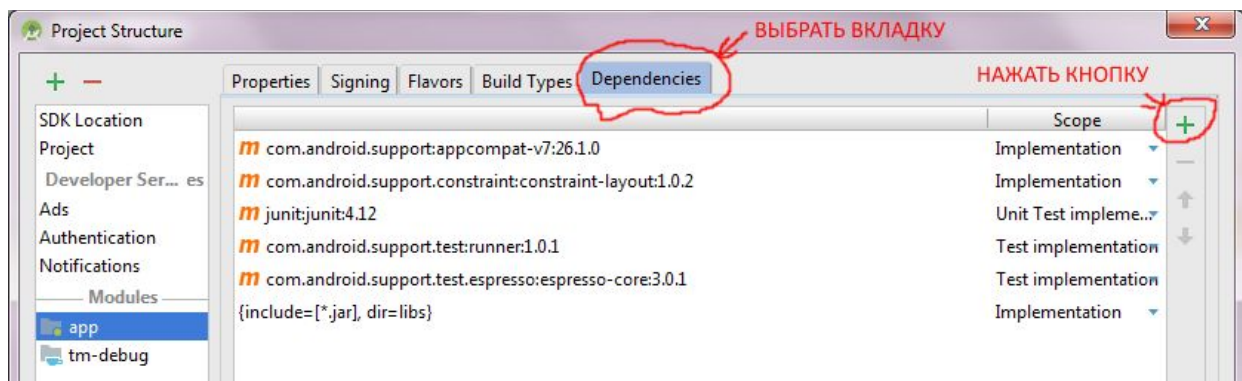
5. В следующем слайде диалога «Create New Module» выбрать файл модуля и нажать кнопку «Finish»



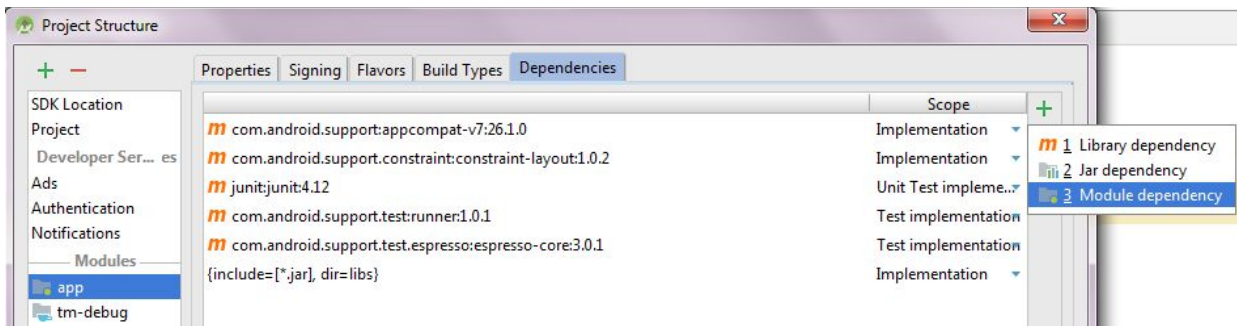
6. В структуре проекта появится модуль для работы с фискальным регистратором:



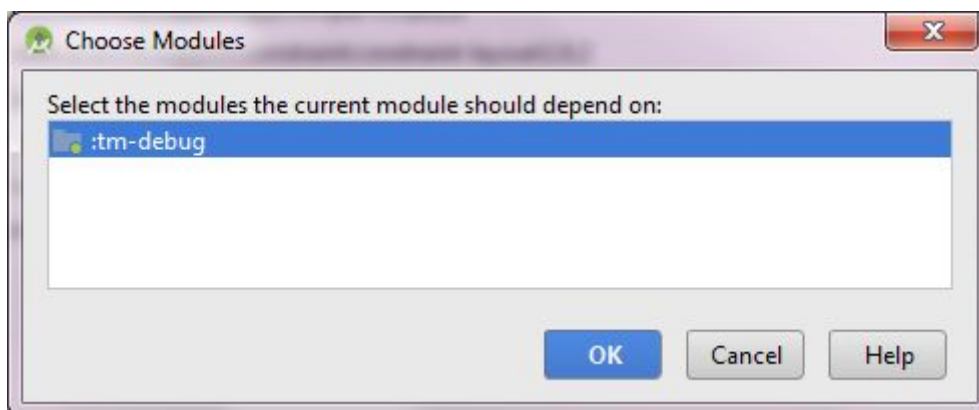
7. Теперь нужно установить зависимости для приложения app. Выбираем в диалоге «Project Structure» модуль app и выбираем вкладку «Dependencies». Затем нажимаем «+».



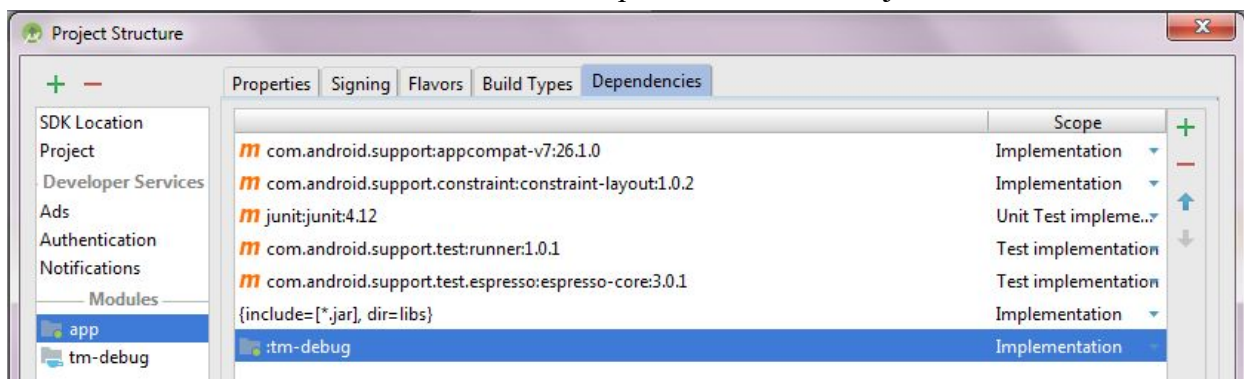
8. В выпадающем меню, выбрать пункт «Module dependency».



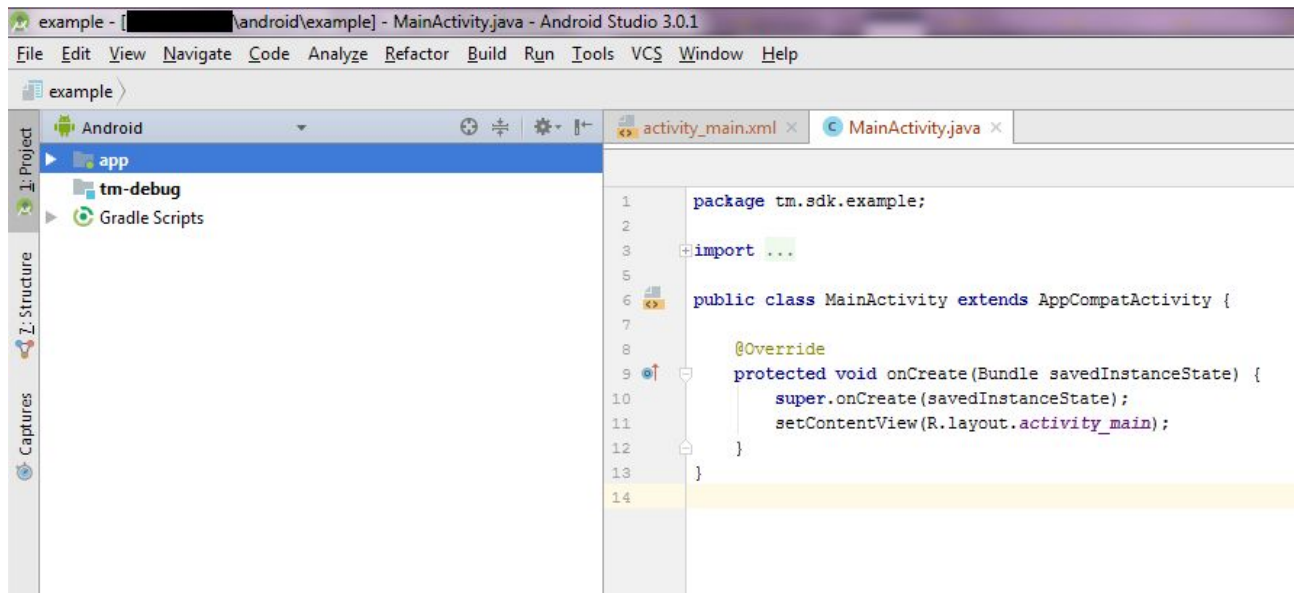
9. Появится диалог «Choose modules», в котором нужно выбрать библиотеку и нажать «OK».



10. После чего появится в диалоге «Project Structure» для модуля app зависимость от библиотеки. Затем нажмите «OK» для закрытия диалога «Project Structure».



После чего можно приступать к использованию модуля в проекте.



5 Краткое описание интерфейсных классов библиотеки

Подробное описание классов приведено в **tm.doc.zip** (сгенерировано через javadoc утилиту из исходных кодов).

В библиотеку входят следующие классы:

- **TMCommand** – класс-контейнер для создания команд контроллеру ФН и распаковки ответов от контроллера ФН. Основной класс для подготовки команд и анализа содержимого ответов от контроллера ФН;
- **TMTag** – класс, перечисление тегов ФФД;
- **TLError** – получения текста ошибки по коду ошибки контроллера ФН;
- **TMDefaults** – определения для работы с сервисом контроллера ФН (типы сообщений и коды ошибок для обмена с андроид-сервисом КФН);
- **TMLib** - класс, для отправки сообщений/команд сервису контроллера ФН и получения ответов от сервиса (определен как singleton);
- **TMLibHandler** – класс, облегчающий расшифровку ответов от **TMLib** (шаблон хэндлера активности для получения ответов);
- **TMOismHandler** – класс, аналогичен **TMLibHandler**, но с дополнительным функционалом: валидации кодов маркировки (для работы по ФФД 1.2 с ТМТ);
- **TMKeysUpdate** - класс, аналогичен **TMLibHandler**, но с дополнительным функционалом: обновление ключей на АС ОКП (для работы по ФФД 1.2 с ТМТ);
- **TMMarkCode** – класс, со вспомогательными функциями для работы с маркировочными кодами;
- **TMOfd** – класс, для фоновой отправки документов в ОФД;
- **TMOism** – класс, для фоновой отправки документов в ОИСМ.

5.1 Обращение к интерфейсным классам модуля

При работе с модулем должны соблюдаться следующие правила:

- Основной интерфейсный класс **TMLib** объявлен как singleton (т. е. единственный экземпляр объекта) и должен существовать все время жизни приложения. Поэтому привязка и инициализация этого класса должна происходить в главном классе приложения (наследованном от **Application**);
- При работе с активностями они должны:
 - При создании (**onCreate**) активность должна создавать хэндлер для приема сообщений от **TMLib** (для этого определен специальный класс **TMLibHandler**, упрощающий обработку сообщений от **TMLib**);
 - При старте (**onStart**) активность регистрирует хэндлер в **TMLib**, с помощью функции **TMLib.registerHandler**;
 - При останове (**onStop**) активность должна сбросить хэндлер в **TMLib**, с помощью функции **TMLib.unregisterHandler**.

5.2 Использование *TMCommand*

Создание новой команды:

- создание контейнера: `TMComand cmd = new TMCommand();`
- создание команды: `cmd.Cmd<название команды>(<параметры команды>);`
- использование команды, например: `tmLib.DoCmd(cmd);`

Пример:

```
TMCommand cmd = new TMCommand();
cmd.CmdGetDocCounters();
libTM.DoCmd(this, cmd, 5000);
```

Распарсивание команды, для распарсивания используются следующие функции:

- `cmd.GetCmdCode()` - получение кода команды;
- `cmd.GetMainError()` - получение основной ошибки;
- `cmd.GetSubError()` - получение дополнительной ошибки;
- `cmd.GetFieldValue(<номер поля>)` - получение значения поля команды (без конвертации);
- `cmd.GetFieldString(<номер поля>)` - получение строки поля команды (с конвертацией из CP866);
- `cmd.GetFieldHexB(<номер поля>)` - получение байтового значения;
- `cmd.GetFieldHexW(<номер поля>)` - получение слова;
- `cmd.GetFieldHexDW(<номер поля>)` - получение двойного слова;
- `cmd.GetFieldBin(<номер поля>)` - получение бинарных данных (с конвертацией из hex представления).

Пример (распарсивание ответа после команды `cmd.FNGetStatus`):

```
text += "\nФН DT: " + cmd.GetFieldValue(10) + "\n";
text += "ФН ЗН: " + cmd.GetFieldValue(11) + "\n";
text += "СРОК ДЕЙСТВ.ФН: " + cmd.GetFieldValue(13) + "\n";
text += "ФН вер: " + cmd.GetFieldString(14) + "\n";
byte typeFN = cmd.GetFieldHexB(15);
text += "ФН тип: " + String.format("x%02X ", typeFN) + ((typeFN == 0) ? "отладочный" : (typeFN == 1)
? "боевой" : "не опр.") + "\n";
byte statFN = cmd.GetFieldHexB(5);
text += String.format(getString(R.string.fn_stage_title)+": x%02X\n", statFN);
if ((statFN & 0x0F) == 1) text += "- "+getString(R.string.fn_stage_1)+"\n";
else if ((statFN & 0x0F) == 3) text += "- "+getString(R.string.fn_stage_2)+"\n";
else if ((statFN & 0x0F) == 7) text += "- "+getString(R.string.fn_stage_3)+"\n";
else if ((statFN & 0x0F) == 15) text += "- "+getString(R.string.fn_stage_4)+"\n";
byte flagsFN = cmd.GetFieldHexB(9);
text += String.format("ФН предупреждения: x%02X\n", flagsFN);
if ((flagsFN & TMCommand.fn_warn_flags.NEED_CHANGE_FN) != 0)
    text += "- до окончания срока действия 3 дня\n";
if ((flagsFN & TMCommand.fn_warn_flags.PREPARE_CHANGE_FN) != 0)
    text += "- до окончания срока действия 30 дней\n";
if ((flagsFN & TMCommand.fn_warn_flags.MEMORY_FULL_FN) != 0)
    text += "- архив ФН заполнен на 90%\n";
if ((flagsFN & TMCommand.fn_warn_flags.WAIT_RESP_OFD) != 0)
    text += "- превышено время ожидания ответа ОФД\n";
```

5.3 Использование TLError

Используется для получения текстовой интерпретации ошибок:

- контроллера ФН;
- системы печати.

5.3.1 Получение текстовой интерпретации ошибок контроллера ФН

В классе TLError определена функция:

```
/**
 * Получение текста ошибки по коду ошибки.
 * @param errorCode код ошибки полученный в ответе ФР.
 * @return текст ошибки.
 */
public static String GetText(byte errorCode)
```

Пример использования функции:

```
if ( cmd.GetMainError() != 0 ) {
    //ошибка при выполнении команды
    MessageBox.Create("ОШИБКА",
        String.format("ОШИБКА ФР (%02X,%02X) ПРИ ИСПОЛНЕНИИ КОМАНДЫ %02X:\n\n",
            cmd.GetMainError(), cmd.GetSubError(), cmd.GetCmdCode()) +
        TLError.GetText((byte)cmd.GetMainError()), MessageBox.option.OK|
        MessageBox.option.WARN).show(getFragmentManager(), "MessageBox");
}
```

5.3.2 Получение текстовой интерпретации ошибок системы печати

В классе TLError определена функция:

```
/**
 * Получение текста ошибки печати по коду ошибки.
 *
 * @param errorCode код ошибки печати.
 * @param addCode дополнительный код ошибки.
 * @return текст ошибки.
 */
public static String GetPrnText(int errorCode, int addCode)
```

Пример использования функции:

```
if (mainErr != 0) {
    MessageBox.Create(getString(R.string.header_error),
        String.format(getString(R.string.msg_print_error), mainErr, subErr)
        +"\n\n"+TLError.GetPrnText(mainErr, subErr),
        MessageBox.option.OK | MessageBox.option.WARN).show(getFragmentManager(),
        "MessageBox");
}
```

5.4 Использование TMLib

TMLib основной интерфейсный класс для работы с сервисом контроллера ФН.

Он включает в себя следующие функции:

- инициализацию работы с интерфейсом сервисом контроллера ФН;
- отправку/прием команд к контроллеру ФН;
- старт/останов фоновых потоков отправки документов в ОФД.

5.4.1 Инициализация класса

Инициализация TMLib рекомендуется производить в классе приложения (наследованного от Application). Так как экземпляр класса используется как singleton (единственный статический) объект и должен существовать все время жизни приложения.

Пример:

```
public class MainApp extends Application {
    /** Интерфейсная библиотека. */
    public static TMLib libTM;
    @Override
    public final void onCreate() {
        super.onCreate();
        //инициализация интерфейсной библиотеки (Singleton)
        libTM = TMLib.getInstance();
        libTM.init(this);
    }
}
```

5.4.2 Работа с контроллером ФН из активностей

При старте активности, нужно регистрировать хендлер (приемщик), для приема сообщений. Для этого нужно переопределить функцию **onStart** активности. Пример:

```
@Override
protected void onStart() {
    super.onStart();
    //получение интерфейса и регистрация хэндлера
    libTM = TMLib.getInstance();
    libTM.registerHandler(h);
}
```

При останове активности, нужно освободить хендлер (приемщик) приема сообщения. Для этого нужно переопределить функцию **onStop** активности. Пример:

```
@Override
protected void onStop() {
    //освобождение хэндлера
    libTM.unregisterHandler(h);
    super.onStop();
}
```

Для обращения к сервису используются следующие функции, определенные в TMLib:

- **Activate** – активация фискального регистратора;
- **DoCmd** – выполнение команды (с возможностью контроля таймаута выполнения, и блокирования интерфейса прогресс-диалогом);
- **DoDLE** – выполнение DLE команды (с возможностью контроля таймаута выполнения, и блокирования интерфейса прогресс-диалогом);

- **SendCmd** – отправка команды (без контроля таймаута);
- **SendDLE** – отправка DLE команды фискальному регистратору;
- **SendNAK** – отправка NAK фискальному регистратору;
- **SendPrintData** – отправка данных для печати;
- **SendPrintParam** – установка параметров печати;
- **SendPrintCut** – отрез печатного документа (если принтер имеет соответствующий функционал);
- **SendRespVer** – отправка запроса на получение версии сервиса.

Для получения ответа от сервиса, нужно определить класс хендлера (**Handler**). Для упрощения обработки получаемых от сервиса сообщений, определен в библиотеке класс-шаблон хендлера **TMLibHandler**.

5.5 Использование TMLibHandler

Класс предназначен для получения сообщений от android-сервиса работающего с КФН.

Содержит следующие шаблоны-функции (приложение должно их переопределить под собственные нужды):

- **onReady** — если связь с андроид-сервисом установлена;
- **onActive** — если ФН активирован (выполнена команда **TMLib.Activate**);
- **onRespCmd** – для получения ответа на команду;
- **onRespData** – для получения данных от КФН (при работе в режиме прямого обмена с КФН);
- **onTimeOut** — в случае превышения времени обращения к КФН (временной интервал задается при вызове функций обращения к КФН в **TMLib**);
- **onError** — в случае ошибки обмена с сервисом КФН;
- **onPrint** — ответ при печати документа через сервис КФН;
- **onVersion** — ответ с информацией о версии сервиса КФН.

Для получения ответных сообщений от android-сервиса работающего с КФН, необходимо переопределить базовый класс **TMLibHandler** для создания собственного хендлера приема ответов от android-сервиса и реализовать перечисленные выше функции (если функция не определена в классе-потомке, то соответствующее сообщения от сервиса будет игнорироваться).

Пример класса-потомка, для обработки сообщений:

```
/**
 * Получение текущих сумм из контроллера ФН.
 *
 * @author <a href="http://www.nilsoft.ru">www.nilsoft.ru</a>, <a
 href="mailto:nilstarsoft@mail.ru">nilstarsoft@mail.ru</a>
 */
public class MoneyActivity extends AppCompatActivity {
    ...
}
```

```

/** Хендлер для обработки сообщений от TMLib. */
private class LibHandler extends TMLibHandler {

    /** Конструктор по умолчанию. */
    public LibHandler() {
        super(MainApp.GetCtx().getMainLooper());
    }

    @Override
    public void onRespCmd(TMCommand cmd) {
        int code = cmd.GetCmdCode();
        switch((byte) cmd.GetCmdCode())
        {
            case TMCommand.commands.CM_GetDateTime: {
                if (cmd.GetMainError() == 0) {
                    FRGetMoney(cmd.GetFieldValue(5), cmd.GetFieldValue(6));
                }
                else FRGetMoney(null, null);
            }
            break;

            case TMCommand.commands.CM_GetMoney: {
                TextView viewText = (TextView) findViewById(R.id.viewMoney);
                if (cmd.GetMainError() != 0) {
                    viewText.append("СВОЙ ВЫПОЛНЕНИЯ КОМАНДЫ:
"+TMErrors.GetText((byte) cmd.GetMainError())+"\n");
                }
                else{
                    //ВЫВОД СЧЕТЧИКОВ
                    ...
                    viewText.setText(text);
                }
            }
            break;
        }
    }

    @Override
    public void onTimeout() {
        //сообщение о превышении таймута
        TextView viewText = (TextView) findViewById(R.id.viewMoney);
        viewText.setText("СВОЙ ВЫПОЛНЕНИЯ КОМАНДЫ: ПРЕВЫШЕН ТАЙМАУТ\n");
    }

    @Override
    public void onError(String errText) {
        //сообщение о сбое сервиса
        TextView viewText = (TextView) findViewById(R.id.viewMoney);
        viewText.setText("СВОЙ РАБОТЫ СЕРВИСА ВЗАИМОДЕЙСТВИЯ С КОНТРОЛЛЕРОМ:
"+errText+"\n");
    }
};
}

```

В примере переопределены функции **onRespCmd**, **onTimeout**, **onError**.

5.5.1 Использование класса TMKeysUpdate (ФФД 1.2 с ТМТ)

Класс включает в себя полностью функционал **TMLibHandler** и дополнительно имеет функционал для обновления ключей на АС ОКП:

- **StartCheck** — проверка необходимости обновления ключей;
- **StartTransfer** – начать обновление ключей.

Этот класс рекомендуется использовать вместо **TMLibHandler** в диалоге открытия смены, для облегчения реализации обновления ключей.

5.5.2 Использование класса **TMOismHandler** (ФФД 1.2 с ТМТ)

Класс включает в себя полностью функционал **TMLibHandler** и дополнительно имеет функционал для валидации маркировочных кодов:

- **StartCheck** — проверка, что необходима валидация маркировочных кодов;
- **StartValidate** – начать валидацию маркировочного кода.

Этот класс рекомендуется использовать вместо **TMLibHandler** в диалоге создания чека, для облегчения реализации валидации маркировочных кодов.

5.6 Использование потока фоновой отправки документов в ОФД

Приложение управляет режимом работы фоновой отправки документов в ОФД.

Внимание! При использовании УПД на программном контроллере ФН, отправка документов производится через него и эти функции не используются (то есть работа через них не производится при включенном УПД).

Рекомендуется запускать поток отправки сообщений в ОФД в следующих случаях:

- Если есть не отправленные документы в ФН;
- После создания нового документа (т. е. после операций открытия/закрытия смены, создания/коррекции чека/БСО и других операций, которые создают документы отчетности).

Для работы с фоновой отправки документов в ОФД используются следующие функции, определенные в **TMLib**:

- **StartOFD** — запуск потока для фоновой отправки документов в ОФД;
- **StopOFD** – останов потока для фоновой отправки документов в ОФД.

5.7 Использование потока фоновой отправки документов в ОИСМ (ФФД 1.2 с ТМТ)

Приложение управляет режимом работы фоновой отправки документов в ОИСМ.

Внимание! При использовании УПД на программном контроллере ФН, отправка документов производится через него и эти функции не используются (то есть работа через них не производится при включенном УПД).

Рекомендуется запускать поток отправки сообщений в ОИСМ в следующих случаях:

- Если есть неотправленные документы в ОИСМ;
- После создания нового документа (т. е. после создания документа с позициями товара, имеющего маркировочный код).

Для работы с фоновой отправки документов в ОИСМ используются следующие функции, определенные в **TMLib**:

- **StartOISM** — запуск потока для фоновой отправки документов в ОИСМ;
- **StopOISM** – останов потока для фоновой отправки документов в ОИСМ.