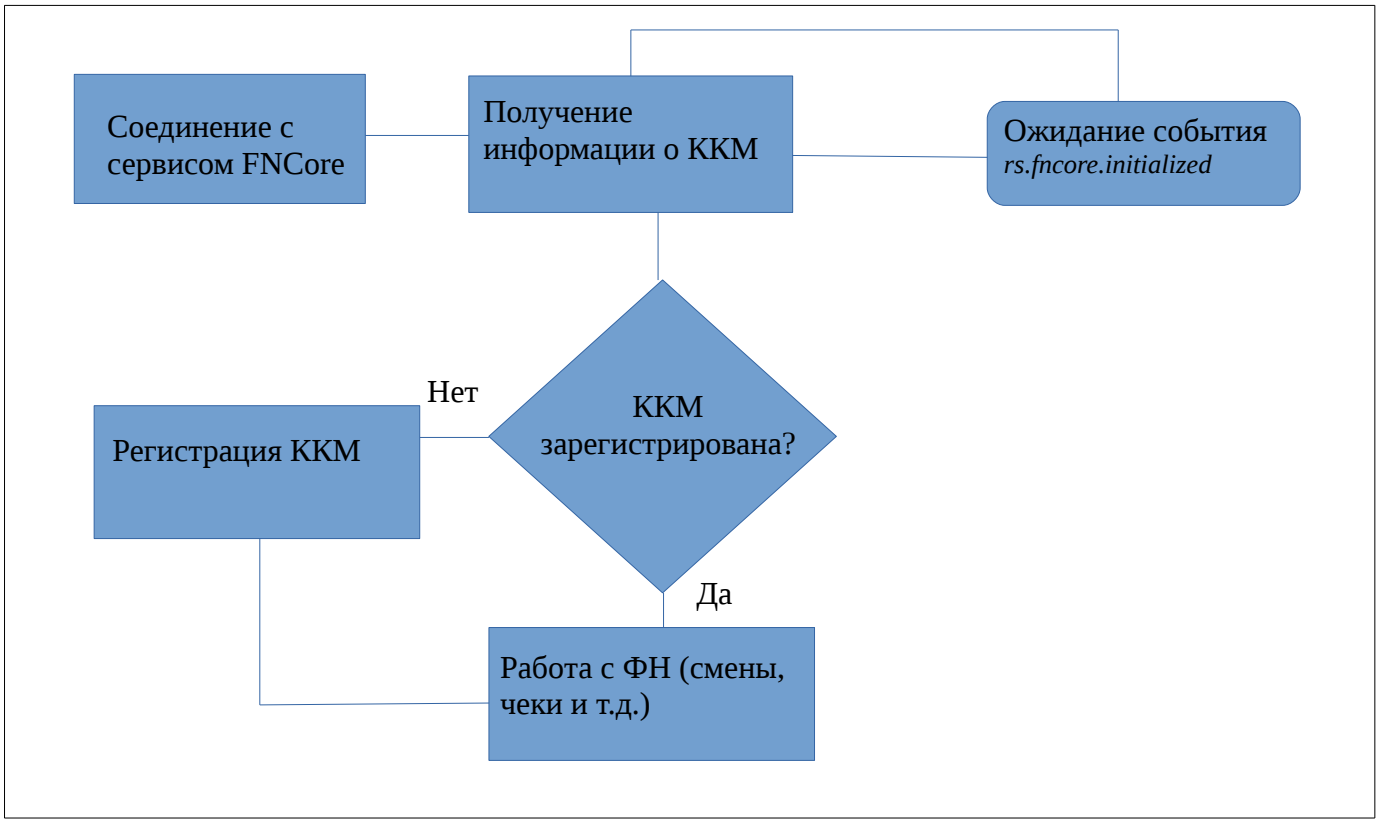
Описание базового API для работы с FNCore

Содержание

Общие принципы взаимодействия с FN-Core

FNCore функционирует как отдельное приложение, взаимодействие с которым осуществляется по IPC как со стандартным service Android. Обратите внимание, что все операции с FNCore могут занимать продолжительное время, поэтому выполнять их в UI-потоке не безопасно. Для вызова методов FNCore следует использовать потоки (в виде Executor, AsynTask и т. д.). Для облегчения работы с FNCore предлагается библиотека FNLibrary, которая реализует основные функции для работы с фискальным накопителем.

Схема взаимодействия с FNCore выглядит следующим образом



Подключение к FNCore

Подключение к FNCore осуществляется стандартным для Android способом, через вызов bindService. Вы можете реализовать это способом описанным ниже.

```
public class FNServiceConnector extends BroadcastReceiver implements
ServiceConnection {
	private Context _context;
	private FiscalService _service;
	public FNServiceConnector() {

	}
	public boolean bindToFNCore(Context context) {
		_context = context;
		if(_context.startService(Const.FN_SERVICE) != null)
			return _context.bindService(Const.FN_SERVICE, this,
Context.BIND_AUTO_CREATE);
		return false;
	}
	public void unbind() {
		if(_service != null)
			_context.unbindService(this);
	}
	@Override
	public void onServiceConnected(ComponentName component, IBinder binder) {
		_service = FiscalService.Stub.asInterface(binder);
		_context.registerReceiver(this, new
IntentFilter(Const.FN_INIT_DONE_ACTION));
	}

	@Override
	public void onServiceDisconnected(ComponentName arg0) {
		_context.unregisterReceiver(this);
		_service = null;
	}
	@Override
```

```
    public void onReceive(Context context, Intent intent) {
    }

}
```

Кроме этого, вам надо зарегистрировать BroadcastReceiver для Action *rs.fncore.initialized*. Это сообщение отправляется всем приложениям, использующим FNCore когда инициализация его завершена и оно готово к работе. После того, как FNCore проинициализирует накопитель, будет разослано событие *rs.fncore.initialized*. Это событие не имеет параметров, и оповещает приложения, что инициализация накопителя завершена. Статус накопителя вы можете получить обычным путем.

Фискальные документы

Любой документ, используемый при работе с FNCore является наследником класса Document. Этот класс описывает документ, позволяющий хранить произвольные данные в формате таг-значение. Этот класс является Parcelable и может быть передан между процессами. Документ имеет следующие поля и методы.

void	**add**(int ID, boolean value) Добавить таг со значением типа boolean (сохраняется как byte 0/1)
void	**add**(int ID, byte value) Добавить таг со значением типа byte
void	**add**(int ID, java.util.Collection<**Tag**> values)
void	**add**(int ID, float value) Добавить тег со значением типа float с 2 знаками после запятой
void	**add**(int ID, float value, int desnsity) Добавить тег со значением типа float с указанным количеством знаков после запятой
void	**add**(int ID, int value) Добавить тег со значением типа int (4 байта)
void	**add**(int ID, short value) Добавить тег со значением типа short (2 байта)
void	**add**(int ID, java.lang.String value) Добавить тег со значением типа строка
void	**add**(int ID, **Tag**... values) Добавить тег со значением типа STLV
void	**cloneTo**(**Document** dest) Скопировать документ
java.lang.String	**getPayAddress**() Получить адрес расчетов
java.lang.String	**getPayPlace**() Получить место расчетов
boolean	**hasTag**(int key)

	Задан ли данный тег для документа
`boolean`	**`isSigned`**`()` Признак "документ подписан"
`void`	**`setPayAddress`**`(java.lang.String value)` Установить адрес расчетов
`void`	**`setPayPlace`**`(java.lang.String value)` Установить место расчетов
`Signature`	**`signature`**`()` Фискальная подпись документа

Получение информации о состоянии ККМ.

Для получения информации о состоянии ККМ используется метод *getKKMInfo()* класса FiscalService. Этот метод возвращает код ошибки, описанный в классе Const.Errors. Если в данный момент выполняется инициализация ФН, то вы получите код возврата INITIALIZATION_IN_PROGRESS и по окончании процесса будет разослано событие *rs.fncore.initialized*, после которого надо будет запросить состояние ККМ повторно. В качестве параметра передается экземпляр класса KKMInfo, этот параметр не может быть null.

Для этой операции возможны следующие коды возвратов и соответствующие операции

Код возврата (константа)	Доступная операция
NO_ERRORS	Все
INITIALIZATION_IN_PROGRESS	Запрос состояния ККМ
NEW_FN	Новый накопитель, доступна регистрация ККМ
DATE_MISMATCH	Неверно установлена дата, операции невозможны
FN_MISMATCH	Установленный ранее ФН не архивирован, установлен новый, операции невозможны
FN_REPLACEMENT	Произведена замена архивированного ФН на новый, доступна перерегистрация ККМ
OLD_FN_HAS_DATA	ФН архивирован, но имеются неотправленные в ОФД документы, доступна ручная отправка одкументов в ОФД
SETTINGS_LOST	Настройки ККМ частично утеряны (сброс ККМ), доступна перерегистрация ККМ с восстановлением настроек
DEVICE_ABSEND	ФН не установлен, операции невозможны

В случае любого ответа (кроме INITIALIZATION_IN_PROGRESS) возвращается заполненный (в некоторых случаях частично) экземпляр класса KKMInfo. Этот класс содержит следующие значимые методы и поля

OU	**casier**() Реквизиты кассира
java.lang.String	**DeviceSerial**() Серийный номер ККТ
java.lang.String	**FNSerial**() Серийный номер ФН
int	**getAgentTypes**() Получить биты агентских услуг (см AGENT_BF_xxx)
int	**getAutomateNumber**() Получить номер автомата
int	**getChecksCount**() Количество чеков за смену
int	**getDocumentsCount**() Количество фискальных документов за смену
java.lang.String	**getFNSUrl**() Получить адрес сайта ФНС
int	**getFNWarnings**() Предупреждения ФН
byte	**getProtocolVersion**() Получить версию протокола ФН/ОФД
java.lang.String	**getRegistrationNo**() Регистрационный номер ККТ
java.lang.String	**getSenderEmail**() Получить e-mail отправителя
byte	**getTaxModes**() Получить битовые флаги СНО (см SNO_BF_xxx)
int	**getUnsavedDocumentType**() Получить тип незавершенного документа
java.lang.String	**getVersion**() Версия ПО ККТ
boolean	**isAutomaticMode**() Получить признак "Установлен в автомате"
boolean	**isBSOMode**() Получить признак "Использование БСО"
boolean	**isCasinoMode**() Получить признак "Проведение азартных игр"
boolean	**isEncryptionMode**() Получить признак "режим шифрования"
boolean	**isExcisesMode**() Получить признак "Продажа подакцизного товара"
boolean	**isFNArchived**() ФН архивный

boolean	**isFNAvailable**() ФН присутствует и доступен для работы
boolean	**isFNClosed**() ФН закрыт
boolean	**isFNHasUnsavedDocument**() Есть незавершенные документы
boolean	**isFNReady**() ФН Фискализирован
boolean	**isInternetMode**() Получить признак "ККТ для Интернет"
boolean	**isLotteryMode**() Получить признак "Проведение лотереи"
boolean	**isOfflineMode**() Получить признак "автономная работа"
boolean	**isServiceMode**() Получить признак "Оказание услуг"
boolean	**isWorkDayOpen**() Признак открытой смены
long	**lastDocumentDate**() Дата последнего фискализированного документа
int	**lastDocumentNumber**() Номер последнего фискализированного документа
OU	**ofd**() Реквизиты ОФД
FNStatistics	**OFDStatistic**() Информация о неотправленных документах
OU	**owner**() Реквизиты владельца ККТ
long	**whenWorkdayisOpen**() Дата открытия последней смены
int	**workDayNumber**() Номер открытой(последней) смены

Регистрация/перерегистрация ККМ

Для регистрации/перерегистрации ККМ используется метод doRegistration() класса FiscalService. Этот метод получает 3 параметра:

Значение параметра	Тип	Коментарии
Причина регистрации	int	Причина регистрации, константы описаны в классе KKMInfo, поля REASON_XX: REASON_REGISTER — регистрация REASON_REPLACE_FN — замена ФН

		REASON_CHANGE_OFD — изменение настроек ОФД REASON_CHANGE_SETTINGS — изменение настроек ККМ и ОФД REASON_CHANGE_KKT_SETTINGS — изменение настроек ККМ
Информация о ККМ	KKMInfo	Запполненные данные о регистрации ККМ
Документ о регистрации ККМ	KKMInfo	Фискализированный документ о регистрации ККМ

В качестве второго и третьего параметра может быть использован один и тот же экземпляр класса KKMInfo. Для регистрации должны быть заполнены следующие значения (минимально)

- Регистрационный номер ККМ
- ИНН и наименование владельца
- ИНН и наименование ОФД (если ККМ находится не в режиме оффлайн)
- Адрес расчетов
- Место расчетов
- Системы налогообложения
- Адрес ФНС в Интернет
- Признак шифрования данных (если ККМ находится не в режиме оффлайн)
- Электронный адрес отправителя

Обратите внимание, что значения передаваемые в методы *setAgentTypes()/setTaxModes()* являются битовыми флагами, и могут быть скомбинированы посредством |.

Метод doRegistration() является ресурсоемким и рекомендуется выполнять его в отдельном потоке. Этот метод может возвращает следующие значения из констант Const.Errors.

Возвращаемое значение	Трактовка
NO_ERRORS	Регистрация выполнена успешно
HAS_UNSET_DOCS	Имеются неотправленные документы в ОФД
0x02	Неверное состояние ФН (есть открытая смена или незавершенный документ)

В случае успешного выполнения операции в возвращаемом экземпляре класса KKMInfo будет выставлен флаг *isSigned()* и заполнены фискальные данные документа возвращаемые методом *signature()*. Этот метод возвращает экземпляр класса Signature который содержит следующие значимые поля и методы.

long	**date**() Дата документа
boolean	**hasOFDReply**() Ответ оператора ОФД получен?
int	**number**() Фискальный номер документа
int	**OFDAnswer**() код ответа оператора ОФД
byte[]	**OFDReply**() Квитанция оператора ОФД
long	**signature**() Фискальная подпись документа

Если вы указываете режим обмена с ОФД, то вам требуется выставить параметры связи с ОФД (адрес сервера, таймаут, стратегию отправки). Это можно выполнить в любой момент времени используя метод *setKKMSettings()* класса FiscalService. Для получения установленных параметров используется метод *getKKMSettings()* того же класса. Методы используют/возвращают экземпляр класса KKMSettings. Его поля описаны ниже

int	**getNetworkTimeout**() Время ожидания ответа от ОФД
int	**getOFDPort**() Порт сервера ОФД
int	**getOFDSendMode**() Получить стратегию отправки документов в ОФД
java.lang.String	**getOFDServer**() Адрес сервера ОФД
int	**getOFDSessionLength**() Получить максимальную длительность сессии отправки документов
void	**setNetworkTimeout**(int value) Установить время ожидания ответа от ОФД
void	**setOFDPort**(int value) Установить порт сервера ОФД
void	**setOFDSendMode**(int v) Установить стратегию отправки документов в ОФД 0 — вручную 1 — по закрытию смены 2 — немедленно
void	**setOFDServer**(java.lang.String value)
void	**setOFDSessionLength**(int v)

	Установить максимальную длительность сессии отправки документов (0 не ограничено)

Таким образом, код для регистрации может выглядеть следующим образом:

```
private void register(final FiscalService service, final Context context) {
            final KKMInfo info = new KKMInfo();
            info.owner().setName("Рога и копыта");
            info.owner().setINN("0123456789");
            info.setPayAddress("г. Москва");
            info.setPayPlace("Центральный офис");
            info.setTaxModes(KKMInfo.SNO_BF_COMMON); // Общая система
налогообложения
            info.setRegistrationNo("0000000001123456");
            info.setFNSUrl("http://nalog.ru");
            info.setSenderEmail("sale@ofice.local");
            info.setEncryptionMode(true);
            info.ofd().setName("ОФД");
            info.ofd().setINN("1234567890");
            new AsyncTask<Void, Void, Integer>() {
                  private ProgressDialog _progress;
                  protected void onPreExecute() {
                        _progress = new ProgressDialog(context);
                        _progress.setIndeterminate(true);
                        _progress.setMessage("Выполняется регистрация ККМ");
                        _progress.show();
                  };

                  @Override
                  protected Integer doInBackground(Void... args) {
                        try {
                              // Установка параметров соединения с ОФД
                              KKMSettings settings = service.getKKMSettings();
                              settings.setOFDPort(7777);
                              settings.setOFDServer("test.ofd.local");
                              settings.setNetworkTimeout(60);
                              settings.setOFDSessionLength(0);
                              settings.setOFDSendMode(1); // Стратегия отправки
документов - по закрытию смены
                              service.setKKMSettings(settings);

                              return
service.doRegistration(KKMInfo.REASON_REGISTER, info, info);
                        } catch(RemoteException re) {
                              return Errors.SYSTEM_ERROR;
                        }
                  }
                  protected void onPostExecute(Integer result) {
                        _progress.dismiss();
                        Toast.makeText(context, String.format("Регистрация
завершена с кодом %02X", result.intValue()),Toast.LENGTH_LONG).show();
                  };
            }.execute();

      }
```

Открытие/закрытие смены

Для операций со сменой предназначены два метода класса FiscalService — openWorkDay() и closeWorkDay(). Оба метода принимают параметр заполненный экземпляр класса OU

содержащий информацию о кассире, выполняющем операцию и заполняют экземпляр класса WorkDay. Этот параметр не может быть null. Основные методы класса WorkDay описаны ниже.

`int`	**`checksCount`**`()` Количество чеков за смену
`int`	**`documentCount`**`()` Количество документов
`boolean`	**`isOpen`**`()` Признак открытой смены
`int`	**`number`**`()` Номер смены
`FNStatistics`	**`OFDStatistic`**`()` Данные о неотправленных документах

Методы могут возвращать следующий результат

Значение	Описание
NO_ERRORS	Операция выполнена успешно
INVALID_WORKDAY_STATE	Смена уже открыта/закрыта
0x02	Неверное состояние ФН (есть незавершенный документ)

В случае успеха заполняется экземпляр класса Signature возвращаемый методом *signature()* в классе WorkDay. Пример использования

```
private void doShift(final boolean open, final FiscalService service, final
Context context) {

            new AsyncTask<Void, Void, Integer>() {
                  private ~~ProgressDialog~~ _progress;
                  private WorkDay _workday = new WorkDay();
                  protected void onPreExecute() {
                        _progress = new ~~ProgressDialog~~(context);
                        _progress.~~setIndeterminate~~(true);
                        _progress.~~setMessage~~("Выполняется операция со сменой");
                        _progress.show();
                  };

                  @Override
                  protected Integer doInBackground(Void... args) {
                        try {
                              OU casier = new OU();
                              casier.setName("Администратор");
                              int result = open ? service.openWorkDay(casier,
_workday) : service.closeWorkDay(casier, _workday);
                              return result;
                        } catch(RemoteException re) {
                              return Errors.SYSTEM_ERROR;
                        }
```

```
                }
                protected void onPostExecute(Integer result) {
                        _progress.dismiss();
                        String toast = String.format("Операция завершена с кодом
%02X", result.intValue());
                        if(result.intValue() == Errors.NO_ERROR)
                                toast +="\nНомер смены "+_workday.number();
                        Toast.makeText(context,toast,Toast.LENGTH_LONG).show();
                };
        }.execute();
    }
```

Отмена незавершенного документа

Если имеется незавершенный документ, то его можно отменить с помощью метода resetDocument() класса FiscalService. Этот метод всегда возвращает NO_ERROR и заполняет передаваемый в него экземпляр класса KKMInfo. Этот параметр не может быть null.

Пример использования

```
private void cancelDocument(FiscalService service) {
        KKMInfo info = new KKMInfo();
        try {
                service.resetDocument(info);
        } catch(RemoteException re) { }
}
```

Формирование отчета о состоянии расчетов

Для формирования отчета о расчетах предназначен метод requestFiscalReport() класса FiscalService. Этот метод может быть вызван при любом состоянии смены, и принимает один параметр — экземпляр класса FiscalReport, который заполняется при успешной операции. Этот класс имеет следующие значимые методы

boolean	**isWDOpen**() Признак открытой смены
FNStatistics	**OFDStatistic**() Информация о неотправленых документах
int	**workDayNumber**() Номер открытой смены

Пример использования метода

```
private void requestReport(final FiscalService service, final Context context) {

        new AsyncTask<Void, Void, Integer>() {
                private ~~ProgressDialog~~ _progress;
                private FiscalReport _report = new FiscalReport();
                protected void onPreExecute() {
                        _progress = new ~~ProgressDialog~~(context);
                        _progress.~~setIndeterminate~~(true);
                        _progress.~~setMessage~~("Выполняется формирование отчета о
расчетах");
                        _progress.show();
                };
```

```
                @Override
                protected Integer doInBackground(Void... args) {
                        try {
                                return service.requestFiscalReport(_report);
                        } catch(RemoteException re) {
                                return Errors.SYSTEM_ERROR;
                        }
                }
                protected void onPostExecute(Integer result) {
                        _progress.dismiss();
                        Toast.makeText(context,String.format("Операция завершена
с кодом %02X", result.intValue()),Toast.LENGTH_LONG).show();
                };
        }.execute();
    }
```

Формирование чека расхода/прихода/возврата

Чек может быть сформирован только при условии что смена открыта и ее длительность не превышает 24 часа. Сам чек описывается экземпляром класса SellOrder. При создании экземпляра класса требуется указать тип документа (приход, расход, возврат прихода, возврат расхода). Для этого предназначены константы TYPE_xxx в классе SellOrder. После требуется задать предметы расчета с помощью метода addItem(). Параметром метода выступает экземпляр класса SellItem. Этот класс содержит следующие поля

int	**AGENT_TYPE** Тип агентской услуги
int	**ITEM_TYPE** Тип предмета расчета
java.lang.String	**NAME** Наименование предмета расчета
int	**PAY_TYPE** Признак способа расчета
float	**PRICE** Стоимость единицы предмета расчета
float	**QTTY** Количество предмета расчета
int	**VAT_TYPE**

Поле ITEM_TYPE определяет тип предмета расчета, для этого используются константы *ITEM_TYPE_xxx* класса SellItem, поле VAT_TYPE определяет ставку налогоболожения (константы *VAT_TYPE_xxx*), поле PAY_TYPE — способ расчета (константы *PAY_TYPE_xxx*). Если предмет расчета содержит признак агента (например оплата в пользу третьего лица) то требуется указать поле AGENT_TYPE (используется **значение** битового флага *AGENT_BF_xxx* из класса KKMInfo, т. е. для указания признака «платежный субагент» используйте значение 1 << KKMinfo.AGENT_BF_PAYMENT_SUBAGENT). Информация об агенте в этом случае задается через установку значения для тега 1224. Пример создания

предмета расчета «Платеж оператору» приведен ниже

```
SellItem item = new SellItem();
item.ITEM_TYPE = SellItem.ITEM_TYPE_PAYMENT; // Тип предмета расчета — платеж
item.PAY_TYPE = SellItem.PAY_TYPE_FULL; // Способ оплаты - полная оплата

item.VAT_TYPE = SellItem.VAT_TYPE_20; // Ставка НДС - 20%
item.AGENT_TYPE = (1 << KKMInfo.AGENT_BF_PAYMENT_AGENT); // Тип агента -
платежный агент
item.QTTY = 1.0f;
item.PRICE = 100.0f;
item.add(1223,
            new Tag(1044, "Платеж"), // тип перевода
            new Tag(1026, "Оператор"), // Оператор перевода
            new Tag(1016, "0123456789") // ИНН оператора перевода
);
```

Таким же образом можно указать (если требуется) агентские данные для самого чека.

Обратите внимание, что размер чека ограничен 32кб (это примерно 40-50 предметов расчета). Этот показатель лучше контролировать при создании чека. После заполнения предметов расчета требуется указать способы оплаты и используемую систему налогообложения. Для этого предназначен метод *setPaymentsDetails()*. Он принимает три параметра — код системы налогообложения (используйте значение констант SNO_BF_xxx класса KKMInfo, т. е. «общая система налогообложения» будет как 1 << KKMInfo.SNO_BF_COMMON), список способов оплаты (List<Payment>) и наименование и ИНН кассира, выполняющего операцию.
Способ оплаты (класс Payment) имеет следующие поля

`float` **SUM**
Сумма оплаты

`int` **TYPE**
Тип оплаты

Где поле TYPE указывает на тип способа оплаты (константа PAYMENT_TYPE_xxx класса Payment). Ниже приведен пример формирования чека прихода

```
SellOrder order = new SellOrder(SellOrder.TYPE_INCOME); // тип чека - приход

SellItem item = new SellItem(); // Создание предмета расчета - товар
item.ITEM_TYPE = SellItem.ITEM_TYPE_GOOD; // Тип предмета расчета - платеж
item.PAY_TYPE = SellItem.PAY_TYPE_FULL; // Способ оплаты - полная оплата
item.VAT_TYPE = SellItem.VAT_TYPE_10; // Ставка НДС - 10%
item.NAME = "Товар";
item.QTTY = 1.0f;
item.PRICE = 200.0f;
order.addItem(item);

item = new SellItem(); // Создание предмета расчета - платеж в пользу третьего
лица
item.ITEM_TYPE = SellItem.ITEM_TYPE_PAYMENT; // Тип предмета расчета - платеж
item.PAY_TYPE = SellItem.PAY_TYPE_FULL; // Способ оплаты - полная оплата
item.VAT_TYPE = SellItem.VAT_TYPE_20; // Ставка НДС - 20%
item.AGENT_TYPE = (1 << KKMInfo.AGENT_BF_PAYMENT_AGENT); // Тип агента -
платежный агент
item.QTTY = 1.0f;
item.PRICE = 100.0f;
item.add(1223,
            new Tag(1044, "Платеж"), // тип перевода
            new Tag(1026, "Оператор"), // Оператор перевода
            new Tag(1016, "0123456789") // ИНН оператора перевода
```

```
);
order.addItem(item);

List<Payment> payments = new ArrayList<>();
Payment payment = new Payment();
payment.TYPE = Payment.PAYMENT_TYPE_CASH; // Наличными 180
payment.SUM = 180.0f;
payments.add(payment);

payment = new Payment();
payment.TYPE = Payment.PAYMENT_TYPE_CARD; // Картой  120
payment.SUM = 120.0f;
payments.add(payment);

OU casier = new OU();
casier.setName("Кассир");
order.setPaymentsDetails((1 << KKMInfo.SNO_BF_COMMON), payments, casier);
```

После этого чек готов к проведению. Проведение чека осуществляется методом *doSellOrder()* класса FiscalService. Этот метод имеет 4 параметра — экземпляр класса SellOrder который требуется провести, экземпляр класса SellOrder который будет заполнен при удачном завершении операции (эти два параметра могут быть одним объектом, но не могут быть null), признак необходимости печати чека, и дополнительный строковый параметр (подвал чека) который может быть напечатан после самого чека (может быть null). Пример проведения чека приведен ниже.

```
private void doSellOrder(final SellOrder order, final FiscalService service,
final Context context) {
            new AsyncTask<Void, Void, Integer>() {
                  private ProgressDialog _progress;
                  protected void onPreExecute() {
                        _progress = new ProgressDialog(context);
                        _progress.setIndeterminate(true);
                        _progress.setMessage("Выполняется проведение чека");
                        _progress.show();
                  };
                  @Override
                  protected Integer doInBackground(Void... args) {
                        try {
                          String footer = "      СКИДКА\n"+
                                          "                    _  ___ _  __     \n"+
                                          "     / |/ _ (_)/ /    \n"+
                                          "     | | | | |/ /     \n"+
                                          "     | | |_| / /_     \n"+
                                          "     |_|\\___/_/(_)    ";
                           return service.doSellOrder(order, order, true,
footer);
                        } catch(RemoteException re) {
                              return Errors.SYSTEM_ERROR;
                        }
                  }
                  protected void onPostExecute(Integer result) {
                        _progress.dismiss();
                        Toast.makeText(context, String.format("Чек проведен с
кодом %02X", result.intValue()),Toast.LENGTH_LONG).show();
                  };
            }.execute();
      }
```

Формирование чека коррекции

Для формирования чека коррекции используется метод *doCorrection()* класса FiscalService. Этот метод использует экземпляр класса Correction для описания проводимой коррекции. Класс содержит следующие методы

OU	**casier**() Реквизиты кассира
java.util.List<**Payment**>	**payments**() Платежи по документу коррекции
void	**setCorrectionType**(int t)
void	**setDocumentNumber**(java.lang.String s) Установить номер документа основания
void	**setSum**(float f) Установить сумму корекции
void	**setType**(int value) Установить тип коррекции
void	**setVATMode**(int type) Установить используемую СНО
void	**setVATType**(int type) Установить ставку налогообложения

Для установки типа коррекции используется константа TYPE_BY_xxx класса Correction. Система налогообложения устанавливается аналогично тому, как это производится при проведении чека. Для установки ставки налогообложения используются константы *VAT_TYPE_xxx* класса SellItem. Ниже приведен пример формирования и проведения произвольного чека коррекции.

```
private void doCorrection(final FiscalService service, final Context context) {
      new AsyncTask<Void, Void, Integer>() {
            private ProgressDialog _progress;
            protected void onPreExecute() {
                  _progress = new ProgressDialog(context);
                  _progress.setIndeterminate(true);
                  _progress.setMessage("Выполняется проведение коррекции");
                  _progress.show();
            };
            @Override
            protected Integer doInBackground(Void... args) {
                  try {
                        Correction correction = new Correction();
                        // Произвольная коррекция
      correction.setCorrectionType(Correction.TYPE_BY_PERCEPT);
                        // Тип коррекции - возврат
                        correction.setType(SellOrder.TYPE_OUTCOME);
                        // Сумма коррекции 300
                        correction.setSum(300f);
                        // Основная СНО
                        correction.setVATMode(1 << KKMInfo.SNO_BF_COMMON);
                        // Без НДС
                        correction.setVATType(SellItem.VAT_TYPE_NONE);
                        // Номер и дата документа-основания проведения коррекции
                        correction.setDocumentNumber("Приказ 11/22");
```

```
                        correction.getDocumentDate().setTime(new Date(2018, 12,
27));

                        // Платеж
                        Payment payment = new Payment();
                        payment.TYPE = Payment.PAYMENT_TYPE_CASH; // Наличными
300
                        payment.SUM = 300f;

                        correction.payments().add(payment);
                        return service.doCorrection(correction, correction);
                  } catch(RemoteException re) {
                        return Errors.SYSTEM_ERROR;
                  }
            }
            protected void onPostExecute(Integer result) {
                  _progress.dismiss();
                  Toast.makeText(context, String.format("Чек коррекции проведен
с кодом %02X", result.intValue()),Toast.LENGTH_LONG).show();
            };
      }.execute();
}
```

Архивирование ФН

Для архивирования ФН предназначен метод *closeFN()* класса FiscalService. Этот метод принимает три параметра — экземпляр класса OU с именем кассира, выполняющего операцию, экземпляр класса ArchiveReport который будет заполнен при успешном выполнении операции и экземпляр KKMInfo который будет заполнен так же при успешном выполнении операции.

Класс ArchiveReport не содержит каких-либо дополнительных полей или методов, и предоставляет информацию о фискальных данных документа.

```
private void doArchive(final FiscalService service, final Context context) {
      new AsyncTask<Void, Void, Integer>() {
            private ProgressDialog _progress;
            protected void onPreExecute() {
                  _progress = new ProgressDialog(context);
                  _progress.setIndeterminate(true);
                  _progress.setMessage("Архивирование ФН");
                  _progress.show();
            };

            @Override
            protected Integer doInBackground(Void... args) {
                  try {
                        OU casier = new OU();
                        casier.setName("Администратор");
                        return service.closeFN(casier, new ArchiveReport(), new
KKMInfo());
                  } catch(RemoteException re) {
                        return Errors.SYSTEM_ERROR;
                  }
            }
            protected void onPostExecute(Integer result) {
                  _progress.dismiss();
                  Toast.makeText(context,String.format("Операция завершена с
кодом %02X", result.intValue()),Toast.LENGTH_LONG).show();
```

```
            };
        }.execute();
}
```

Реестр проведенных операций

FNCore ведет реестр проведенных операций, которые доступны через механизм ContentProvider по адресу *content://rs.fncore.data/documents*. Этот провайдер предоставляет следующие поля

Наименование поля	Тип данных	Описание
DOCNO	INTEGER	Фискальный номер документа
DOC_TYPE	INTEGER	Тип документа
BODY	BLOB	Тело документа
CC	INTEGER	Код ответа от ОФД
SUBTITLE	TEXT	Читаемое обозначение типа документа
DOC_DATE	INTEGER	Дата документа в милисекундах, unixtime

Этот провайдер предназначен только для чтения, и хранит данные только по активному фискальному накопителю. Что бы получить сохраненный документ в виде экземпляра класса-наследника Document нужно использовать метод *unmarshall()* класса Utils.
Значение поля DOC_TYPE описано константами в классе Const.DocTypes.

Реестр сменных остатков

Реестр сменных остатков является вспомогательным реестром, который фиксирует суммы по фискальным операциям. Он реализован в виде механизма Content Provider и доступен по адресу *content://rs.fncore.data/rests.* Этот провайдер предоставляет следующие поля

Наименование поля	Тип данных	Описание
DOCNO	INTEGER	Фискальный номер документа
PAY_TYPE	INTEGER	Тип платежа
SUM	REAL	Сумма платежа со знаком (положительные — приходные операции, отрицательные — расходные операции)

Тип платежа соответствует значению констант *PAYMENT_TYPE_xxx* в класса Payment.
Этот провайдер доступен как для чтения так и для записи, т. е. Вы можете хранить в нем нефискальные операции прихода/расхода. В этом случае значение поля DOCNO нужно формировать программисту. Данные в этом реестре очищаются при операции открытия смены.
Если вам требуется внести или изъять наличные, то вы можете выполнить вставку для этого провайдера. В этом случае, поле DOCNO требуется заполнять самому, рекомендуется использовать для этого время операции. Пример работы с этим Content Provider приведен ниже

```
private boolean nonFiscalCorrection(KKMInfo info, OU ou, Context ctx, float sum)
{
      Uri RESTS = Uri.parse("content://rs.fncore.data/rests");
      ContentValues cv = new ContentValues();
      cv.put("DOCNO", System.currentTimeMillis()/1000);
```

```
	cv.put("PAY_TYPE",Payment.PAYMENT_TYPE_CASH);
	cv.put("SUM",sum);
	if(ctx.getContentResolver().insert(RESTS, cv) != null) {
		IPrintManager pm =  PrinterManager.getInstance();
		int y = pm.drawText(Utils.align("ККТ", info.getRegistrationNo()), 0,
0);
		y += pm.drawText(Utils.align("ИНН", info.owner().getINN()), 0, y);
		if(ou.isSet())
			y += pm.drawText(Utils.align("Кассир",ou.getName()),0,y);
		y +=
pm.drawText(Utils.align(Utils.formateDate(System.currentTimeMillis()),
				ou.getName()), 0, y);
		y += pm.drawText(" ", 0, y);
		y += pm.drawText(Utils.center(sum > 0 ? "ВНЕСЕНИЕ" : "ИЗЯТИЕ"), 0,
y, true);
		y += pm.drawText(Utils.right(String.format("=%.2f", Math.abs(sum))),
0, y);
		for (int i = 0; i < 5; i++)
			y += pm.drawText(" ", 0, y);
		pm.printPage();
		pm.close();
		return true;
	}
	return false;
}
```

Печать нефискальных отчета по сменным остаткам

FNCore поддерживает печать нефискального отчета по сменным остаткам. Для этого предназначен метод *printXReport()* класса FiscalService. Этот метод не имеет входных параметров и всегда возвращает NO_ERROR.

Отправка документов в ОФД

Для принудительной отправки документов в ОФД используется метод *sendDocuments()* класса FiscalService. Этот метод выполняется достаточно долго, поэтому его требуется вызывать в отдельном потоке. Метод принимает один параметр — экземпляр класса KKMInfo который будет заполнен после вызова функции. Он не может быть null.
Пример использования метода

```
private void doSendDocuments(final FiscalService service, final Context context)
{
	new AsyncTask<Void, Void, Integer>() {
		@Override
		protected Integer doInBackground(Void... args) {
			try {
				return service.sendDocuments(new KKMInfo());
			} catch(RemoteException re) {
				return Errors.SYSTEM_ERROR;
			}
		}
	}.execute();
}
```

Сброс фискального накопителя

Для МГМ доступна функция сброса накопителя. Он осуществляется через метод *reset()* класса FiscalService. Этот метод принимает на вход экземпляр класса KKMInfo, который

будет заполнен при удачном выполнении метода. Метод должен вызываться только в отдельном потоке.