Getnet

SDK de Hardware

POS Digital | Getnet

Documentação

Versão 2.1.1

CLASSIFICAÇÃO DO DOCUMENTO – CONFIDENCIAL

Sumário

Histórico de alterações

Versão	Data	Descrição
2.0	13/05/2020	Ajustes na organização da documentação.
2.1	26/02/2021	Melhorias e correções.
2.1.1	06/05/2021	Melhoria: Boas práticas de uso do SDK.

SDK de Hardware

O SDK Getnet é uma camada de abstração entre o app e o *hardware* disponibilizado através de serviços dentro do Android. A utilização dele é obrigatória, pois isso garante que o aplicativo irá funcionar em todos modelos de POS Digital da Getnet. O SDK é composto por 3 arquivos:

Você pode baixar estes arquivos no Portal do Desenvolvedor, no menu “Documentação e SDKs”.

servicePosDigital.apk: é o serviço do SDK que fica rodando no terminal que se comunica com os recursos de *hardware*. No Portal do Desenvolvedor, está com o nome SDK de Hardware (Aplicativo).

libPosDigital.aar: é a interface do SDK que deve ser compilado com o seu app para se comunicar com o serviço. No Portal do Desenvolvedor, está com o nome SDK de Hardware (Aplicativo).

Devkit.apk: é um app de ajuda que acessa os serviços e tem alguns exemplos. Com ele, o desenvolvedor consegue validar se os serviços de *hardware* estão funcionando corretamente. No Portal do Desenvolvedor, está com o nome Devkit.

IMPORTANTE: Instale o servicePosDigital.apk antes de qualquer outro aplicativo.

IMPORTANTE: A Getnet se reserva ao direito de solicitar ao desenvolvedor que adapte o seu aplicativo quando houver alguma mudança no SDK. Métodos deprecados serão mantidos somente durante uma versão.

Abaixo estão os serviços disponíveis e como eles estão em cada hardware.

Serviço	Ingenico – APOS A8	Newland – N910
Card	Suporta	Suporta
Printer	Suporta	Suporta
Mifare	Suporta	Suporta
Leds	Suporta	Suporta
Beeper	Suporta	Suporta
Câmera (Leitor de barcode e qrcode)	Suporta	Suporta

Configurar Gradle

Importar a lib “libposdigital.aar” no Gradle e a classe *PosDigital* estará disponível para uso.

No *gradle* do seu app adicione este trecho:

```
android {
    compileSdkVersion 27
    defaultConfig{
        minSdkVersion 22
        targetSdkVersion 27
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.aar'])
}
```

Fazendo bind no service

Adicione no AndroidManifest.xml a permissão abaixo:

```
<uses-permission android:name="com.getnet.posdigital.service.POSDIGITAL" />
```

O **Posdigital.register()** é melhor utilizado no contexto da Aplicação ou dentro de um BaseActivity onde todas as Activitys da aplicação irão herdar os métodos da BaseActivity. Somente após o método onConnected ser executado que os serviços irão funcionar.
No código exemplo abaixo, o register é feito no BaseActivity:

```
abstract class BaseActivity : AppCompatActivity() {

    private val tag = javaClass.name

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.content_activity)
        connectPosDigitalService()
    }

    private fun connectPosDigitalService() {
        PosDigital.register(applicationContext, bindCallback)
    }

    override fun onDestroy() {
        super.onDestroy()
        try {
            if (PosDigital.getInstance().isInitiated)
                PosDigital.unregister(applicationContext)
        } catch (e: Exception) {
            Log.e(tag, "Erro de exception no Destroy da Activity")
        }
    }

    private val bindCallback: PosDigital.BindCallback
        get() = object : PosDigital.BindCallback {
            override fun onError(e: Exception) {
               if (PosDigital.getInstance().isInitiated)
                   PosDigital.unregister(applicationContext)

                connectPosDigitalService()
             }
            override fun onConnected() {}

            override fun onDisconnected() {}
        }
}
```

Caso ocorra a desconexão do serviço no callback (**onDisconnected**) recomendamos:

- Exibir uma mensagem na tela permitindo o usuário fazer uma nova tentativa de conexão;
- Ou então fazer uma nova tentativa em background, mas por um número máximo de vezes - recomendamos 3 tentativas.

IMPORTANTE: Cada vez que o serviço de hardware do fabricante parar de funcionar é emitido um evento, este evento é recebido no onError() do bindCallback. É importante implementar a reconexão com o SDK como mostrado no exemplo acima, no método onError() do código.

A Activity que herdará da BaseActivity cada vez que irá fazer alguma operação com o SDK, deverá fazer a verificação ***PosDigital.getInstance().isInitiated*** para verificar se o SDK está inicializado, conforme código exemplo abaixo:

ActivityImpressão.kt

```
fun printReceipt() {
   if (PosDigital.getInstance().isInitiated){
       try {
         PosDigital.getInstance().printer.init()
         PosDigital.getInstance().printer.setGray(5)
         PosDigital.getInstance().printer.defineFontFormat(FontFormat.MEDIUM)
         PosDigital.getInstance().printer.addText(AlignMode.LEFT, "Barcode:
20")
         PosDigital.getInstance().printer.addText(AlignMode.LEFT, "    ")
         PosDigital.getInstance().printer.print(getPrinterCallback())
       } catch (e: Exception) {}
   }else{
     // falha no service de impressão
     // refaça o start da activity que
     // extende a BaseActivity para reconectar o service
     openErrorDialog("Falha na impressão\n Tente novamente.")
   }
}
private fun getPrinterCallback(): IPrinterCallback.Stub {
    return object : IPrinterCallback.Stub() {
        @Throws(RemoteException::class)

        override fun onSuccess() {
            openInfoDialog("Impresso com sucesso")
        }

        @Throws(RemoteException::class)

        override fun onError(cause: Int) {
            openErrorDialog(parseStatus(cause))
        }
    }
}
```

IMPORTANTE: Não utilizar o PosDigital.register() em uma thread separada, como em uma AsyncTask() do android, pode gerar problemas de concorrência de Threads e trazer instabilidade e erros nas chamadas ao SDK.

Card

Para utilizar os eventos de cartões basta chamar o método PosDigital.getInstance().getCard() e estão disponíveis as funções das leitoras. Exemplo:

```
int timeout = 30; //segundos
int[] searchType = {SearchType.MAG, SearchType.CHIP, SearchType.NFC};

PosDigital.getInstance().getCard().search(timeout, searchType, new ICardCallback.Stub(){

    @Override
    public void onCard(CardResponse cardResponse){}

    @Override
    public void onMessage(String message){}

    @Override
    public void onError(String error){}

});
```

Impressora

Para utilizar a impressora basta chamar o método *PosDigital.getInstance().getPrinter()* e estão disponíveis as funções da impressora.
Exemplo:

```
PosDigital.getInstance().getPrinter().init();
PosDigital.getInstance().getPrinter().setGray(5);
PosDigital.getInstance().getPrinter().defineFontFormat(FontFormat.MEDIUM);
PosDigital.getInstance().getPrinter().addText(AlignMode.LEFT, "EXEMPLO DE TEXTO");
PosDigital.getInstance().getPrinter().print(new IPrinterCallbackStub() {

    @Override
    public void onSuccess(){
        //Imprimiu o texto
    }

    @Override
    public void onError(int cause){
        //Ocorreu algum erro
    }

});
```

Descrição das funções de impressão

init()

Inicializa a impressora no *hardware*, não requer parâmetros adicionais.

setGray(int gray)

Seta escalas de cinza na impressão, requer um parâmetro INT.

defineFontFormat(int fontFormat)

Define o tamanho da fonte que vai ser impresso, requer um parâmetro INT(FontFormat.SMALL, FontFormat.MEDIUM, FontFormat.LARGE).

addText(int align, String text)

Imprime texto, mas não imprime caracteres especiais. Recebe dois(2) parâmetros, um é o align(LEFT, CENTER ou RIGTH), e a string de texto. Máximo de caracteres por linha em todos os terminais:

- SMALL - 48 caracteres
- MEDIUM - 32 caracteres
- LARGE - 32 caracteres

A função addText() adiciona automaticamente um "\n" no final. Se utilizar apenas um addText() para todo o comprovante, deve ser inserido manualmente o "\n" pelo programador, respeitando o limite máximo de caracteres por linha. Se este limite não for respeitado, **o wordwrap default** poderá causar desalinhamento no comprovante. **Caso utilize addText() para inserir linhas em branco para corte de papel, recomenda-se inserir um espaço vazio, para compatibilidade(addText(AlignMode.LEFT, " ")).**

Também deve-se respeitar o número máximo de caracteres por addText(), que é 1800 caracteres, caso este limite não seja respeitado, poderá haver problemas de não impressão de comprovantes.

addImageBitmap(int align, Bitmap bitmap)

Adiciona imagens em formato bitmap para impressão a partir da pasta Assets ou Resource. O método possui uma limitação na largura do BMP. No máximo **378** pixels de largura para a imagem poder ser alinhada (esquerda, direita, centralizado). Imagens com largura maior que **378** pixels são impressas sempre centralizadas. Imagens com largura acima de **378** pixels serão redimensionadas para **378** pixels de largura sem distorção da imagem. Portanto é recomendado usar imagens com no máximo **378** pixels de largura.

addBarCode(int align, String barcode)

Adiciona um barcode para ser impresso, recebendo os parâmetros align(int) e barcode(String). Para a legibilidade e qualidade do barcode, **é recomendado utilizar o**

número máximo de 28 caracteres, caso possua entre 28 e 60 caracteres será feito um resize automático e será impresso com uma qualidade inferior. Se possuir mais de 60 caracteres, poderá não ser impresso.

addQrCode(int align, int imageheight, String qrcode)

Adiciona um QrCode para ser impresso, recebendo os parâmetros align(int), altura em pixels(int) e qrCode(String). Para a legibilidade e qualidade do QrCode, **é recomendado uma altura em pixels de no mínimo 140, e máximo 300. Fora deste range, será feito um resize automático do QrCode para os valores citados acima.**

addImageByteArray() [DEPRECATED]

Adiciona imagens em formato byteArray para impressão a partir da pasta Assets ou Resource. **A função está descontinuada, e não deverá mais ser utilizada.**

print(in IPrinterCallback callback)

Realiza a impressão, requer parâmetro getPrinterCallBack().

printAndRemovePaper(in IPrinterCallback callback)

Realiza a impressão com um espaçamento no final para o texto visível sair da impressora, permitindo seu corte. Requer parâmetro getPrinterCallBack().

getStatus() – Retorna um int com o status da impressão.

IMPORTANTE: sempre agrupe as informações no comprovante antes de mandar imprimir, ou seja, chame o método "*print*" após todo o comprovante já estar pronto. Sempre que chamar o método "*print*", aguarde os eventos de call-back antes de chamar outro método "*print*".

O tratamento de erros da impressão pode ser feito com o auxílio das constantes da classe *PrinterStatus*.

```
PrinterStatus.OK -> "OK"
PrinterStatus.PRINTING -> "Imprimindo"
PrinterStatus.ERROR_NOT_INIT -> "Impressora não iniciada"
PrinterStatus.ERROR_OVERHEAT -> "Impressora superaquecida"
PrinterStatus.ERROR_BUFOVERFLOW -> "Fila de impressão muito grande"
PrinterStatus.ERROR_PARAM -> "Parâmetros incorretos"
PrinterStatus.ERROR_LIFTHEAD -> "Porta da impressora aberta"
PrinterStatus.ERROR_LOWTEMP -> "Temperatura baixa demais para impressão"
PrinterStatus.ERROR_LOWVOL -> "Sem bateria suficiente para impressão"
PrinterStatus.ERROR_MOTORERR -> "Motor de passo com problemas"
PrinterStatus.ERROR_NO_PAPER -> "Sem bobina"
PrinterStatus.ERROR_PAPERENDING -> "Bobina acabando"
PrinterStatus.ERROR_PAPERJAM -> "Bobina travada"
PrinterStatus.UNKNOW -> "Não foi possível definir o erro"
```

```
public class PrinterStatus {
    OK = 0;
    PRINTING = 1;
    ERROR_NOT_INIT = 2;
    ERROR_OVERHEAT = 3;
    ERROR_BUFOVERFLOW = 4;
    ERROR_PARAM = 5;
    ERROR_LIFTHEAD = 10;
    ERROR_LOWTEMP = 11;
    ERROR_LOWVOL = 12;
    ERROR_MOTORERR = 13;
    ERROR_NO_PAPER = 15;
    ERROR_PAPERENDING = 16;
    ERROR_PAPERJAM = 17;
    UNKNOW = 1000;
}
```

Dica de impressão

Para imprimir com acentos e caracteres especiais, é bom criar um .xml no seu código, com o esqueleto da impressão, populá-lo, e após utilizar a função de converter para Bitmap, como no exemplo abaixo:

```
val view = layoutInflater.inflate(R.layout.slip, null)
PosDigital.getInstance().getPrinter().addImageBitmap(AlignMode.CENTER, ViewUtils.convertToBitmap(view))
```

Exemplo de XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="378px"
   android:layout_height="wrap_content"
   android:orientation="vertical">

   <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">

      <android.support.v7.widget.AppCompatTextView
         style="@style/TextAppearance.Slip"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_alignParentStart="true"
         android:layout_marginStart="22dp"
         android:text="Posto de Combustível XYZ" />

   </RelativeLayout>

   <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content">

      <android.support.v7.widget.AppCompatTextView
         style="@style/TextAppearance.Slip.Small"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_alignParentStart="true"
         android:layout_marginStart="37dp"
         android:text="CNPJ: 00.666.000/0001.66" />

   </RelativeLayout>
```

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <android.support.v7.widget.AppCompatTextView
        style="@style/TextAppearance.Slip.Medium"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_marginStart="25dp"
        android:text="MASTER/VISA/AMEX/ELO" />

</RelativeLayout>

<android.support.v7.widget.AppCompatTextView
    style="@style/TextAppearance.Slip.Medium"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="DEBITO/CREDITO PARCELADO/A VISTA"
    android:textAlignment="center" />

<android.support.v7.widget.AppCompatTextView
    style="@style/TextAppearance.Slip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="R$ 12,34"
    android:textAlignment="center" />

</LinearLayout>
```

Mifare

Para utilizar o Mifare basta chamar o método PosDigital.getInstance().getMifare() e estão disponíveis as funções do Mifare.

O Mifare na Getnet é amplamente usado nas automações, e os cartões que são conhecidos pela empresa são os Mifare Classic, de 1k e 4k(S50, S70, S50 PRO, S70 PRO).

Descrição das funções do Mifare **(representada na linguagem Kotlin)**

activate(cardType: Int): Int

Ativa o cartão Mifare para transação.

halt()

Interrompe todos os comandos Mifare.

getCardSerialNo(cardType : Int): String

Ativa e retorna o UID do cartão aproximado.

searchCard(callback: IMifareCallback)

Retorna o tipo de cartão Mifare que foi aproximado.

searchCardAndActivate(callback: IMifareActivateCallback)

Procura cartão na proximidade e ativa o cartão aproximado.

authenticateSectorWithKeyA(index: Int, key: ByteArray): Int

Autentica no setor especificado com a chave A.

authenticateBlockWithKeyA(index: Int, key: ByteArray): Int

Autentica no bloco especificado com a chave A.

authenticateSectorWithKeyB(index: Int, key: ByteArray): Int

Autentica no setor especificado com a chave B.

authenticateBlockWithKeyB(index: Int, key: ByteArray): Int

Autentica no bloco especificado com a chave B.

close()

Desliga a antena contactless.

decrement(index: Int, value: Int): Int

Decrementa o conteúdo de um bloco e armazena o resultado no Buffer interno.

increment(index: Int, value: Int): Int

Incrementa o conteúdo de um bloco e armazena o resultado no Buffer interno.

isExist(): Boolean

Retorna se existe um cartão próximo a antena ou não.

readBlock(index: Int): String

Lê um bloco em específico.

restore(index: Int): Int

Move informações de um bloco para o buffer.

transfer(index: Int): Int

Transfere informações do buffer interno para um bloco.

writeBlock(index: Int, data: String): Int

Escreve em um bloco específico.

exchangeAPDU (apduIn: ByteArray): APDUResponse

Processa troca de dados em protocolo ISO14443-4.

Exemplos de códigos Mifare

Exemplo de uma leitura de bloco

```
private fun readBlock() {
  val dialogBuilder = AlertDialog.Builder(this)
  val inflater = this.layoutInflater
  val dialogView = inflater.inflate(R.layout.custom_dialog, null)
  dialogBuilder.setView(dialogView)

  val etBlock = dialogView.findViewById<EditText>(R.id.block)
  val etKey = dialogView.findViewById<EditText>(R.id.key)
  val llIncrement = dialogView.findViewById<LinearLayout>(R.id.llIncrement)
  val llData = dialogView.findViewById<LinearLayout>(R.id.llData)
  val keyType = dialogView.findViewById<Spinner>(R.id.keyType)

  llData.visibility = View.GONE
  llIncrement.visibility = View.GONE

  dialogBuilder.setTitle("Parâmetros")
  dialogBuilder.setMessage("Informe o setor, bloco e a chave")

  dialogBuilder.setPositiveButton("OK") { dialog, whichButton ->
     val auxBlock = etBlock.text.toString().trim()
     val auxKey = etKey.text.toString().trim()

     if (!auxBlock.isEmpty() && !auxKey.isEmpty()) {
        block = auxBlock.toInt()
        key = hexStringToByte(auxKey)

        val callback = openActionDialog("Mifare", "Aproxime o cartão ...")

        waitCard(object : MifareCallback {
           override fun onSuccess(type: Int) {
```

```
            val activateResponse = PosDigital.getInstance().mifare.activate(type)

            if (activateResponse != MifareStatus.SUCCESS) {
                callback.onFinish("Não foi possível ativar o dispositivo")
                return
            }

            val authSectorResponse = if(keyType.selectedItem.equals("A"))
            PosDigital.getInstance().mifare.authenticateSectorWithKeyA(block /4, key)
            else
            PosDigital.getInstance().mifare.authenticateSectorWithKeyB(block /4, key)

            if (authSectorResponse != MifareStatus.SUCCESS) {
                callback.onFinish("Não foi possível autenticar o setor")
                return
            }

            val authBlockResponse = if(keyType.selectedItem.equals("A"))
                PosDigital.getInstance().mifare.authenticateBlockWithKeyA(block, key)
            else
                PosDigital.getInstance().mifare.authenticateBlockWithKeyB(block, key)

            if (authBlockResponse != MifareStatus.SUCCESS) {
                callback.onFinish("Não foi possível authenticar o bloco")
                return
            }

            val result = PosDigital.getInstance().mifare.readBlock(block)
            if(result == null){
                callback.onFinish("Não foi possível ler o bloco com esta chave de autenticação")
                return
            }

            callback.onFinish("Read [$result]")
        }

        override fun onError(message: String) {
            callback.onFinish(message)
        }
    })
    } else {
        Toast.makeText(this, "Todos os parâmetros precisam ser preenchidos", Toast.LENGTH_SHORT).show()
    }
}
dialogBuilder.create().show()
PosDigital.getInstance().mifare.halt()
}
```

Exemplo de uma escrita no bloco

```
private fun writeBlock() {
    val dialogBuilder = AlertDialog.Builder(this)
    val inflater = this.layoutInflater
    val dialogView = inflater.inflate(R.layout.custom_dialog, null)
    dialogBuilder.setView(dialogView)

    val etBlock = dialogView.findViewById<EditText>(R.id.block)
    val etKey = dialogView.findViewById<EditText>(R.id.key)
    val llIncrement = dialogView.findViewById<LinearLayout>(R.id.llIncrement)
    val etData = dialogView.findViewById<EditText>(R.id.data)
    val keyType = dialogView.findViewById<Spinner>(R.id.keyType)

    llIncrement.visibility = View.GONE

    dialogBuilder.setTitle("Parâmetros")
```

```
    dialogBuilder.setMessage("Informe o setor, bloco, chave e os dados")

    dialogBuilder.setPositiveButton("OK") { dialog, whichButton ->
        val auxBlock = etBlock.text.toString().trim()
        val auxKey = etKey.text.toString().trim()
        val auxData = etData.text.toString().trim()

        if (!auxBlock.isEmpty() && !auxKey.isEmpty()&& !auxData.isEmpty()) {
            val data = auxData
            block = auxBlock.toInt()
            key = hexStringToByte(auxKey)

            val callback = openActionDialog("Mifare", "Aproxime o cartão ...")

            waitCard(object : MifareCallback {
                override fun onSuccess(type: Int) {

                    val activateResponse = PosDigital.getInstance().mifare.activate(type)

                    if (activateResponse != MifareStatus.SUCCESS) {
                        callback.onFinish("Não foi possível ativar o dispositivo")
                        return
                    }

                 val authSectorResponse = if(keyType.selectedItem.equals("A"))
                  PosDigital.getInstance().mifare.authenticateSectorWithKeyA(block/4,key)
                    else
                  PosDigital.getInstance().mifare.authenticateSectorWithKeyB(block/4,key)

                    if (authSectorResponse != MifareStatus.SUCCESS) {
                        callback.onFinish("Não foi possível authenticar o setor")
                        return
                    }

                    val authBlockResponse = if(keyType.selectedItem.equals("A"))
                     PosDigital.getInstance().mifare.authenticateBlockWithKeyA(block,key)
                    else
                     PosDigital.getInstance().mifare.authenticateBlockWithKeyB(block,key)

                    if (authBlockResponse != MifareStatus.SUCCESS) {
                        callback.onFinish("Não foi possível authenticar o bloco")
                        return
                    }

                val writeResponse = PosDigital.getInstance().mifare.writeBlock(block,data)

                    if (writeResponse != MifareStatus.SUCCESS) {
                        callback.onFinish("Não foi possível escrever no bloco com esta chave de autenticação")
                        return
                    }

                    val result = PosDigital.getInstance().mifare.readBlock(block)

                    callback.onFinish("write [$result]")
                }

                override fun onError(message: String) {
                    callback.onFinish(message)
                }
            })
        } else {
            Toast.makeText(this, "Todos os parâmetros precisam ser preenchidos", Toast.LENGTH_SHORT).show()
        }
    }
    dialogBuilder.create().show()
    PosDigital.getInstance().mifare.halt()
}
```

IMPORTANTE: enviar previamente as tags Mifare para a Getnet. Solicitar por e-mail o endereço. Sem as tags, não será possível efetuar os testes e a aplicação será reprovada.

Exemplo de uso exchangeAPDU

Este método disponibiliza a interface ISO14443-4 para o desenvolver acessar cartões sem além do Mifare, por exemplo Cipurse.

```
PosDigital.getInstance().mifare.halt()
PosDigital.getInstance().mifare.searchCard(object : IMifareCallback.Stub() {
    override fun onCard(type: Int) {
        try {
            when (type) {
                MifareType.PRO_CARD, MifareType.S50_PRO_CARD, MifareType.S70_PRO_CARD -> {
                    PosDigital.getInstance().mifare.close()
                    PosDigital.getInstance().mifare.activate(MifareType.PRO_CARD)
                    val res = PosDigital.getInstance().mifare.exchangeAPDU("00A40000022FF7".toByteArray())
                    Log.d(TAG, "ret:${res.apduRet} sw1:${res.SW1.toHexStr()} sw2:${res.SW2.toHexStr()}")
                }
                else -> Log.e(TAG, "Cartao desconhecido")
            }
        } catch (e: RemoteException) {
            Log.e(TAG, "Exception reading on ingenico: ${e.localizedMessage}")
        }
    }
    override fun onError(error: String) {
        Log.e(TAG, "Exception reading on ingenico: ${error}")
    }
})
```

Leds

Para ligar e desligar os Leds basta chamar o método PosDigital.getInstance().getLeds() e estarão disponíveis as funções dos lds.

```
PosDigital.getInstance().getLed().turnOnAll()
PosDigital.getInstance().getLed().turnOffAll()
```

Beeper

Para utilizar o beeper basta chamar o método PosDigital.getInstance().getBeeper() e estão disponíveis as funções do *beeper*.

```
PosDigital.getInstance().getBeeper().success()
```

Câmera

Para utilizar a câmera basta chamar o método PosDigital.getInstance().getCamera() e estão disponíveis as funções da câmera.

```
int timeout = 30; //segundos
PosDigital.getInstance().getCamera().readBack(timeout, new ICameraCallback.Stub(){

    @Override
    public void onSuccess(String code){}

    @Override
    public void onTimeout(){}

    @Override
    public void onCancel(){}

    @Override
    public void onError(String error){}

});
```

Info

Para obter as informações de versão do SDK, do SO, do número de série e da biblioteca compartilhada usar o método:
PosDigital.getInstance().getInfo().info(callbackInfo()).

Como pode ser visto no exemplo abaixo as informações estão disponíveis no *callback* onInfo no objeto InfoResponse.

```
try{
    PosDigital.getInstance().getInfo().info(new IInfoCallback.Stub(){

    @Override
    public void onInfo(InfoResponse infoResponse) throws RemoteException{
        StringBuilder info = new StringBuilder()
        .append("SDK Version:[").append(infoResponse.getSdkVersion()).append("]\n")
        .append("BC Version:[").append(infoResponse.getBcVersion()).append("]\n")
        .append("OS Version:[").append(infoResponse.getOsVersion()).append("]\n")
        .append("Serial Number:[").append(infoResponse.getSerialNumber()).append("]");
    }

    @Override
    public void onError(String s) throws RemoteException{
        Log.e(TAG, String.format("onError %s", s));
    }

    });

} catch(RemoteException e){
    e.printStackTrace();
}
```

Estatística

Para consultar as estatísticas do terminal, tais como:

- Consumo de bobina
- Leitura de cartão magnético (tarja)
- Leitura de cartão EMV contato (chip)
- Leitura de cartão EMV sem contato (Contactless)
- Leitura de cartão Mifare

O serviço Getnet possui um conjunto de métodos, divididos em ***estatística geral*** e ***estatística por apps*** e as estatísticas de leitura de cartão possuem funções de leitura com sucesso e falha.

Estatística Geral

```
PosDigital.getInstance().getStatistic().getPaperStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getMifareStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getMifareStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getMagStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getMagStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getChipStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getChipStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getNfcStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getNfcStatusFail(IStatCallback callback)
```

Estatística por App

```
PosDigital.getInstance().getStatistic().getAllPaperStatus(IStatCallback.Stub callback)
PosDigital.getInstance().getStatistic().getAllAppMagStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllAppMagStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllAppChipStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllAppChipStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllAppNfcStatus(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllAppNfcStatusFail(IStatCallback callback)
PosDigital.getInstance().getStatistic().getAllStatisticsByApp(IStatCallback callback)
```

A informação é obtida através do *callback* **IStatCallback**. O exemplo mostra como obter a estatística do consumo de bobina do terminal.

```
PosDigital.getInstance().getStatistic().getPaperStatus(new IStatCallback.Stub(){

    @Override
    public void onStatistic(StatResponse statResponse) throws RemoteException{
        Log.i(TAG, String.format("Consumo de bobina (Geral): %s", statResponse.getGeneralPaperStatus()));
    }

    @Override
    public void onError(String s) throws RemoteException{
        Log.e(TAG, String.format("Erro: %s", s));
    }
});
```