



[Manual de Integração Android Moderninha Smart](#)

Sumário

Versionamento deste documento	1
Introdução	4
Observações Importantes	5
Importando a biblioteca PlugPagService Wrapper	6
AndroidManifest.xml	7
Permissões	7
Intent-filter	7
Classes	8
Interfaces	11
API	12
PlugPag	12
Constantes	12
Construtores	14
Métodos	14
PlugPagAbortResult	25
Construtores	25
Métodos	25
PlugPagActivationData	25
Construtores	25
PlugPagAppIdentification	26
Construtores	26
Métodos	26
PlugPagBeepData	26

Constantes	27
Constructores	28
PlugPagException	28
Constructores	28
PlugPagInitializationResult	29
Constructores	29
PlugPagEventData	29
Constantes	30
Constructores	33
Métodos	33
Métodos estáticos	34
PlugPagLedData	35
Constantes	35
Constructores	36
PlugPagNFCAuth	36
Constructores	36
PlugPagNearFieldCardData	37
Constantes	37
PlugPagSimpleNFCDData	39
Constructores	39
PlugPagNFCInfosResult	39
Constructores	39
PlugPagPaymentData	40
Constructores	40

Métodos.....	41
PlugPagPaymentData.Builder	42
Constructores	42
Métodos.....	42
PlugPagTransactionResult.....	45
Constructores	45
Métodos.....	46
PlugPagTransactionResult.Builder	50
Constructores	50
Métodos.....	50
PlugPagVoidData.....	54
Constructores	54
Métodos.....	55
PlugPagVoidData.Builder	56
Constructores	56
Métodos.....	56
PlugPagCustomPrinterLayout	57
Constructor.....	57
Métodos.....	57
PlugPagNFCResult	58
Constructor.....	58
Métodos.....	58
PlugPagPrintResult.....	60
Constructor.....	60

Métodos.....	60
PlugPagPrinterData	60
Construtor.....	60
Métodos.....	61
PlugPagMockResult.....	61
Construtor.....	61
Constantes	61
PreferredNetwork	65
Constantes	65
TerminalCapabilities	66
Constantes	66
PlugPagPrinterListener.....	68
Métodos.....	68
PlugPagEventListener.....	69
Métodos.....	69
PlugPagPaymentListener	69
Métodos.....	69
PlugPagNFCListener	70
Métodos.....	70
PlugPagLastTransactionListener	71
Métodos.....	71
PlugPagIsActivatedListener.....	71
Métodos.....	71
PlugPagInstallmentsListener	72

Métodos.....	72
PlugPagActivationListener	72
Métodos.....	72
PlugPagAbortListener	73
Métodos.....	73
PlugPagCardInfoResult.....	73
Atributos	73
PlugPagTransactionHistory	67
Métodos.....	67
Exemplos	81
Pagamentos	82
Estornar um pagamento	86
Verificar autenticação	87
Invalidar autenticação.....	88
Solicitar ativação	89
Obter versão da biblioteca.....	90
Customizar dialog de impressão da via do cliente	94
Exemplos assíncronos	111
Pagamentos.....	111
Estornar um pagamento.....	119
Verificar autenticação	121
Invalidar autenticação	122
Solicitar ativação	123
Códigos de retorno	133

Códigos de erro de impressão	137
------------------------------------	-----

Versionamento deste documento

Versão doc.	Data	Autor	Descrição	Versão PlugPag Service
1.0.0	28/01/2019	Carlos Vaccari	Criação do documento.	1.0
1.0.1	06/03/2019	Cássio Corrêa	Ajuste layout da capa	1.0
1.0.2	06/03/2019	Cássio Corrêa	Movido seção observações	1.0
1.0.3	14/03/2019	Hugo Yamashita	Correção de exemplos	1.0.7
1.0.4	15/03/2019	Hugo Yamashita	Revisão do documento	1.0.7
1.0.5	19/03/2019	Carlos Vaccari	Adição de NFC	1.1.0
1.0.6	21/03/2019	Carlos Vaccari	Adição de dialog de impressão customizável	1.0.7
1.0.7	05/04/2019	Carlos Vaccari	Correção de exemplo. Adição de observação.	1.1.0
1.0.8	08/04/2019	Carlos Vaccari	Adição de impressão livre.	1.1.0
1.0.9	11/04/2019	Hugo Yamashita	Inclusão da classe PlugPagNearFieldCardData	1.1.1
1.0.10	11/04/2019	Lucas Miranda	Adição dos eventos de digitação de PIN na PlugPagEventData	1.2.0

1.0.11	14/05/2019	Carlos Vaccari	Inclusão de exemplo de busca de última transação	1.3.0
1.0.12	28/05/2019	Carlos Vaccari	Inclusão de abort do NFC	1.3.3
1.0.13	01/11/2019	Carlos Vaccari	Inclusão de novos códigos de eventos	1.5.1
1.0.14	01/11/2019	Carlos Vaccari	Inclusão de métodos assíncronos no wrapper	1.6.0
1.0.15	27/11/2019	Lucas Miranda	Atualizando listeners, data, terminal capabilities, preferred network, custom printer, activation data	1.6.0
1.0.16	02/12/2019	Carlos Vaccari	Adicionando listeners dos métodos assíncronos	1.6.0
1.0.17	16/12/2019	Hugo Yamashita	Revisão	1.6.0
1.0.18	20/03/2020	Henrique Jardim	Adicionado Mock	1.8.0
1.0.19	28/05/2020	Anderson Peres	Inclusão da classe CardData e do método GetCard	1.8.0
1.0.20	26/06/2020	Anderson Peres	Inclusão do QRCode	1.8.2
1.0.21	06/10/2020	Fabiano Arantes	Correção do documento: tag de permissão	1.10.0

1.0.22	19/10/2020	Rodrigo Garcete	Inclusão da classe PlugPagTransactionHistory	1.11.0
1.0.23	10/11/2020	Lucas Miranda	Inclusão pagamento PIX	1.11.1
1.0.24	22/03/2021	Igor R. Bagliotti	Alteração TransactionHistory e ServiceTransactionHistory	1.13
1.0.25	27/04/2021	Igor R. Bagliotti	Remoção do TransactionHistory e Inclusão pré-autorização	1.14
1.0.26	19/05/2021	Igor R. Bagliotti	Separar chamadas NFC (detect card, detect remove e auth)	1.15
1.0.27	03/08/2021	Igor R. Bagliotti	Inclusão documentação chamada NFC APDU	1.16
1.0.28	11/08/2021	Igor R. Bagliotti	Inclusão pagamento QR Code Crédito Elo	1.17

Introdução

Este documento destina-se a integradores que utilizarão os terminais da linha **Smart** do PagSeguro como solução de pagamento integrada através do serviço **PlugPagService**.

O serviço **PlugPagService** pode ser acessado diretamente ou por meio da biblioteca **PlugPagService Wrapper**. Este documento destina-se à explicação do uso da biblioteca **PlugPagService Wrapper**.

Observação:

A linha **Smart** de terminais atualmente é composta dos terminais **Moderninha Smart** e **Moderninha X**. O serviço **PlugPagService** e a biblioteca **PlugPagService Wrapper** são iguais para os modelos.

Observações Importantes

A biblioteca **PlugPagService** para o sistema operacional Android possui algumas restrições para seu uso.

- A biblioteca PlugPag possui suporte da API level 25 (7.1.1 Nougat) à 28 (9.0 Pie), devido às versões de Android presentes nos terminais **Moderninha Smart** e **Moderninha X**.
- Apenas uma única instância do PlugPag deve existir durante o uso do aplicativo. A existência de múltiplas instâncias pode fazer com que o comportamento seja indeterminado.
- As chamadas dos métodos da classe `PlugPag` devem ser feitas em uma `Thread` que execute em background pois podem demorar para finalizar a execução. Caso a execução seja feita na `Thread` principal (`UI Thread`), o aplicativo pode apresentar um ANR (Application Not Responding). Além disso, alguns métodos executam transações utilizando chamadas remotas pela internet, o que impossibilita suas chamadas na `Thread` principal.
- Eventos que chamem duas ou mais vezes o serviço de pagamento ou estorno antes da operação ser finalizada, podem ocasionar comportamento anormal no serviço e requerer que a aplicação seja fechada para realizar o *unbind* do serviço.

Importando a biblioteca PlugPagService Wrapper

Para importar a biblioteca **PlugPagService Wrapper** na sua aplicação nativa Android basta seguir os passos descritos abaixo:

1- Inserir no arquivo *build.gradle* do projeto a URL do repositório Maven do PlugPag:

```
allprojects {
    repositories {
        ...
        maven {
            url 'https://github.com/pagseguro/PlugPagServiceWrapper/raw/master'
        }
        ...
    }
}
```

2- Inserir as dependências no arquivo *build.gradle* da aplicação:

```
dependencies {
    ...
    implementation 'com.android.support:design:28.0.0'
    implementation 'br.com.uol.pagseguro.plugpagservice.wrapper:wrapper:1.6.0'
    ...
}
```

A versão da dependência `com.android.support:design` deve ser a mesma utilizada para as demais dependências `com.android.support`. A versão `28.0.0` é a mais recente no momento da edição desse documento.

AndroidManifest.xml

Permissões

Para integrar a biblioteca *PlugPagService* em aplicativos para Android é necessário adicionar a seguinte permissão ao *AndroidManifest.xml*.

```
<uses-permission android:name="br.com.uol.pagueseguro.permission.MANAGE_PAYMENTS"/>
```

Essa permissão permite à biblioteca realizar o bind ao **PlugPagService**, serviço embarcado do terminal, que gerencia todas as transações de pagamento.

Intent-filter

Para que seu aplicativo possa ser escolhido como aplicativo padrão de pagamento e receber *Intents* de inserção de cartão, é necessário adicionar o seguinte código em seu *AndroidManifest.xml* dentro da sua *Activity* principal.

```
<intent-filter>
    <action android:name="br.com.uol.pagueseguro.PAYMENT"/>
    <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
```

Classes

A biblioteca **PlugPagService** é composta de um conjunto de classes.

A classe principal chama-se `PlugPag`, mas é necessário utilizar classes auxiliares para configurações e trocas de informações.

Segue abaixo uma lista com classes que compõem a biblioteca.

Classe	Descrição
PlugPag	Classe principal da biblioteca. Essa classe é responsável pelas transações.
PlugPagAbortResult	Resultado obtido ao solicitar um cancelamento de operação, enquanto a operação está em andamento.
PlugPagActivationData	Dados de ativação do terminal.
PlugPagApplIdentification	Identificação do aplicativo.
PlugPagBeepData	Configuração de beep.
PlugPagCustomPrinterLayout	Classe para customização da dialog de impressão da via do cliente.
PlugPagEventData	Dados de eventos gerados durante transações para atualização de eventos no aplicativo.
PlugPagException	Tipo principal de exceções geradas pelo PlugPag.
PlugPagInitializationResult	Resultado da inicialização do terminal
PlugPagLedData	Configuração de LED.

PlugPagNearFieldCardData	Dados para leitura/escrita de NFC.
PlugPagNFCAuth	Dados de autenticação do NFC.
PlugPagNFCInfosResult	Resultado de uma requisição para obter informações de NFC.
PlugPagNFCResult	Resultado de uma leitura/escrita NFC.
PlugPagPaymentData	Informações de um pagamento a ser realizado.
PlugPagPrinterData	Informações de uma impressão a ser realizada.
PlugPagPrintResult	Resultado de uma requisição de impressão.
PlugPagSimpleNFCDData	Dados para escrita em um cartão NFC.
PlugPagTransactionResult	Resultado de uma transação.
PlugPagVoidData	Informações de um estorno a ser realizado.
PreferedNetwork	Enum com configurações de conexão (dados móveis).
TerminalCapabilities	Constantes de funcionalidades do terminal.
PlugPagMockResult	Resultado da simulação escolhida.
PlugPagTransactionResult	Resultado de transações de venda e estorno com sucesso.
PlugPagPreAutoData	Dados para criar uma pré-autorização
PlugPagEffectuatePreAutoData	Dados para efetivar uma pré-autorização criada
PlugPagCmdExchangeResult	Dados do retorno do comando APDU (retorna o comando de resposta)

Interfaces

As interfaces visam facilitar e padronizar algumas chamadas de métodos de forma assíncrona.

Interface	Descrição
PlugPagAbortListener	Interface com métodos que são chamados durante uma transação assíncrona de abort.
PlugPagActivationListener	Interface com métodos que são chamados durante uma ativação e/ou durante uma desativação assíncrona.
PlugPagInstallmentsListener	Interface com métodos que são chamados ao realizar o cálculo de parcelas de forma assíncrona.
PlugPagIsActivatedListener	Interface com métodos que são chamados durante uma verificação se está ou não ativado assíncrona.
PlugPagLastTransactionListener	Interface com métodos que são chamados ao tentar obter de forma assíncrona a última transação aprovada.
PlugPagNFCListener	Interface com métodos que são chamados durante uma leitura ou uma escrita assíncrona em um cartão NFC.
PlugPagPaymentListener	Interface com métodos que são chamados durante uma transação assíncrona de pagamento.
PlugPagEventListener	Interface com método chamado quando um evento é enviado durante uma transação.
PlugPagPrinterListener	Interface com métodos chamados quando um evento é enviado durante uma impressão.

API

Abaixo segue a descrição da interface pública da biblioteca **PlugPagService**.

PlugPag

Essa é a classe principal da biblioteca.

É por meio dessa classe que é possível realizar transações nos terminais.

Constantes

int	RET_OK
	Código utilizado para indicar sucesso nas operações.
	Valor: 0
int	REQUEST_CODE_AUTHENTICATION
	Código utilizado para iniciar a <code>Activity</code> de autenticação.
	Valor: 46981
int	TYPE_CREDITO
	Tipo de pagamento: crédito.
	Valor: 1
int	TYPE_DEBITO
	Tipo de pagamento: débito.
	Valor: 2

int	TYPE_VOUCHER	Tipo de pagamento: voucher (vale refeição).	Valor: 3
int	TYPE_QRCODE	Tipo de pagamento: qrcode elo.	Valor: 4
int	TYPE_PIX	Tipo de pagamento: qrcode pix.	Valor: 5
int	TYPE_QRCODE_CREDITO	Tipo de pagamento: qrcode crédito	Valor: 7
int	INSTALLMENT_TYPE_A_VISTA	Forma de parcelamento: à vista.	Valor: 1
int	INSTALLMENT_TYPE_PARC_VENDEDOR	Forma de parcelamento: parcelamento vendedor.	Valor: 2
int	ERROR_REQUIREMENTS_MISSING_PERMISSIONS	Código de retorno para indicar erro de falta de permissões do aplicativo.	Valor: -3000

int ERROR_REQUIREMENTS_ROOT_PERMISSION

Código de retorno para indicar que o aparelho possui permissões de root.

Valor: -3001

Construtores

`PlugPag(Context context, PlugPagAppIdentification appIdentification)`

Cria uma instância do `PlugPag` utilizando `context` para acessar dados e recursos do dispositivo e identificando as transações com os dados do aplicativo fornecidos em `appIdentification`.

Gera uma exceção se `context` ou `appIdentification` forem nulos.

Métodos

Tipo de retorno	Método e descrição
PlugPagAbortResult	<code>abort()</code> Solicita o cancelamento da operação atual. O cancelamento da transação não ocorre instantaneamente, pois depende do fluxo da transação. Retorna o resultado da solicitação de cancelamento.
PlugPagNFCResult	<code>abortNFC()</code> Aborta uma operação de leitura/escrita NFC.

Int authNFCCardDirectly(authData: PlugPagNFCAuth)

Realiza a autenticação do sistema NFC usado.

Retorna sucesso com código 1 e falha com código -1.

void	asyncAbort(listener: PlugPagAbortListener)
	Solicita o cancelamento da operação atual de forma assíncrona.

void	asyncAbortNFC(listener: PlugPagAbortListener)
	Aborta uma operação de leitura/escrita NFC de forma assíncrona.

void	asyncCalculateInstallments(saleValue: String, listener: PlugPagInstallmentsListener)
	Calcula o valor das parcelas de forma assíncrona.

void	asyncDeactivate(activationData: PlugPagActivationData, listener: PlugPagActivationListener)
	Realiza a desativação do terminal de forma assíncrona.

void	asyncGetLastApprovedTransaction(listener: PlugPagLastTransactionListener)
	Obtém a última transação aprovada de forma assíncrona.

void	asyncIsAuthenticated(isActivatedListener: PlugPagIsActivatedListener)
	Verifica se há um usuário autenticado de forma assíncrona.

void	<p>asyncReadNFC(cardData: PlugPagNearFieldCardData, listener: PlugPagNFCListener)</p> <p>Realiza leitura do conteúdo de um cartão NFC de forma assíncrona.</p>
void	<p>asyncReprintCustomerReceipt(listener: PlugPagPrinterListener)</p> <p>Realiza a reimpressão da via do cliente de forma assíncrona.</p>
void	<p>asyncReprintEstablishmentReceipt(listener: PlugPagPrinterListener)</p> <p>Realiza a reimpressão da via do estabelecimento de forma assíncrona.</p>
void	<p>asyncWriteNFC(cardData: PlugPagNearFieldCardData, listener: PlugPagNFCListener)</p> <p>Realiza escrita em um cartão NFC de forma assíncrona.</p>
int	<p>beep(beepData: PlugPagBeepData)</p> <p>Toca um beep</p> <p>Retorna sucesso com código 1 e falha com código -1.</p>
String[]	<p>calculateInstallments(saleValue: String)</p> <p>Calcula o valor das parcelas.</p> <p>Retorna os valores das parcelas.</p>

PlugPagTransactionResult	doPayment(PlugPagPaymentData paymentData) Efetua um pagamento. Retorna o resultado da transação.
PlugPagInitializationResult	deactivate(activationData: PlugPagActivationData) Realiza a desativação do terminal. Retorna o resultado de uma desativação.
void	disposeSubscriber() Realiza o dispose do subscriber atual.
void	doAsyncInitializeAndActivatePinpad(activationData: PlugPagActivationData, listener: PlugPagActivationListener) Realiza a inicialização e a ativação do terminal para uso através do código de ativação de forma assíncrona.
void	doAsyncVoidPayment(voidData: PlugPagVoidData, listener: PlugPagPaymentListener) Efetua um estorno de forma assíncrona de um pagamento identificado pelos dados contidos em voidData.
PlugPagApplIdentification	getApplIdentification() Retorna a identificação do aparelho definido no construtor da classe.
String	getApplicationCode() Retorna o código da aplicação. Esse código é uma constante da biblioteca.

PlugPagTransactionResult	<p>getLastApprovedTransaction()</p> <p>Obtém a última transação aprovada.</p>
String	<p>getLibVersion()</p> <p>Retorna a versão da biblioteca PlugPagService.</p>
PlugPagNFCInfosResult	<p>getNFCInfos(cardType: Int)</p> <p>Obtém as informações de um cartão NFC</p> <p>Retorna o as informações de cartão NFC</p>
Boolean	<p>hasCapability(capability: Int)</p> <p>Verifica se o terminal tem uma funcionalidade específica.</p> <p>Retorna <code>true</code> se houver a funcionalidade, <code>false</code> caso contrário.</p>
int	<p>initBTConnection(PlugPagDevice deviceInformation)</p> <p>Configura a conexão bluetooth utilizando os dados de <code>deviceInformation</code>.</p> <p>Retorna <code>PlugPag.RET_OK</code> em caso de sucesso.</p>
PlugPagInitializationResult	<p>initializeAndActivatePinpad(activationData: PlugPagActivationData)</p> <p>Realiza a inicialização e a ativação do terminal para uso através do código de ativação.</p> <p>Retorna o resultado da inicialização.</p>
void	<p>invalidateAuthentication()</p> <p>Invalida uma autenticação. Equivalente a realizar um logout.</p>

boolean	<p>isAuthenticated()</p> <p>Verifica se há um usuário autenticado.</p> <p>Retorna <code>true</code> se houver um usuário autenticado, <code>false</code> caso contrário.</p>
PlugPagPrintResult	<p>printFromFile(printerData: PlugPagPrinterData)</p> <p>Solicita a impressão de um arquivo.</p>
PlugPagPrintResult	<p>reprintCustomerReceipt()</p> <p>Solicita a reimpressão da via do cliente.</p>
PlugPagPrintResult	<p>reprintStablishmentReceipt()</p> <p>Solicita a reimpressão da via do estabelecimento.</p>
void	<p>requestAuthentication(PlugPagAuthenticationListener listener)</p> <p>Solicita autenticação. O resultado da autenticação é notificado ao listener que é passado no parâmetro <code>listener</code>.</p>
void	<p>setEventListener(PlugPagEventListener listener)</p> <p>Armazena a referência de uma instância de interface que receberá os eventos gerados durante as transações.</p>
Int	<p>setLed(ledData: PlugPagLedData)</p> <p>Define os LEDs que serão acesos.</p> <p>Retorna sucesso com código 1 e falha com código -1.</p>

void	<p><code>setPlugPagCustomPrinterLayout(PlugPagCustomPrinterLayout)</code>.</p> <p>Permite customizar elementos da dialog de impressão da via do cliente.</p>
Boolean	<p><code>setPreferredNetwork(preferedNetwork: Int)</code></p> <p>Configura a tipo de rede Preferido com <code>1</code> (4G/3G/2G), <code>2</code> (3G/2G), <code>3</code> (2G)</p> <p>Retorna sucesso com <code>true</code> e falha com <code>false</code>.</p>
void	<p><code>setPrinterListener(listener: PlugPagPrinterListener)</code></p> <p>Armazena a referência de uma instância de interface que receberá os eventos gerados durante uma impressão.</p>
int	<p><code>setVersionName(String appName, String appVersion)</code></p> <p>Define o nome e a versão do aplicativo que está integrando com o <code>PlugPagService</code>.</p> <p><code>appName</code> pode ter no máximo 25 caracteres.</p> <p><code>appVersion</code> pode ter no máximo 10 caracteres.</p> <p>Retorna um código de erro se um dos parâmetros for nulo ou vazio.</p>
Int	<p><code>startNFCCardDirectly()</code></p> <p>Inicia a antena NFC para uso.</p> <p>Retorna sucesso com código <code>1</code> e falha com código <code>-1</code>.</p>

Int	<code>stopNFCCardDirectly()</code> Finaliza o sistema de NFC em uso. Retorna sucesso com código 1 e falha com código -1.
PlugPagNFCResult	<code>readFromNFCCard(PlugpagNearFieldCardData cardData)</code> Realiza leitura do conteúdo de um cartão NFC. Retorna sucesso com código 1 e falha com código -1.
Int	<code>readNFCCardDirectly(cardData: PlugPagSimpleNFCDData)</code> Realiza leitura do conteúdo de um cartão NFC diretamente. Retorna sucesso com código 1 e falha com código -1.
PlugPagTransactionResult	<code>voidPayment(PlugPagVoidData voidData)</code> Efetua um estorno de um pagamento identificado pelos dados contidos em <code>voidData</code> . Retorna o resultado da transação.
PlugPagNFCResult	<code>writeToNFCCard(PlugpagNearFieldCardData cardData)</code> Realiza escrita em um cartão NFC. Retorna sucesso com código 1 e falha com código -1.
Int	<code>writeToNFCCardDirectly(cardData: PlugPagSimpleNFCDData)</code> Realiza escrita em um cartão NFC diretamente. Retorna sucesso com código 1 e falha com código -1.

Boolean	<p>getMockState(mock: Boolean)</p> <p>True - Ativa o fluxo de mock.</p> <p>False - Desativa o fluxo de mock.</p> <p>Retorna true se mock foi atualizado com sucesso.</p>
Boolean	<p>getMockState()</p> <p>Retorna true estado do mock estiver ativado ou false para o fluxo do mock desativado</p>
Boolean	<p>getMockResult(typeMock: Int)</p> <p>Escolhe o retorno simulado esperado typeMock é o caso de mock selecionado</p> <p>Retorna se o caso de mock foi alterado com sucesso</p>
PlugPagMockResult	<p>getMockResult()</p> <p>Retorna o caso de mock escolhido.</p>
PlugPagTransactionResult	<p>getPreAutoData()</p> <p>Realiza a busca de uma pré-autorização através do cartão do usuário</p> <p>Retorna o resultado da criação da pré-autorização</p>
PlugPagTransactionResult	<p>doPreAutoCancel(transactionId : String, transactionCode : String)</p> <p>Realiza ao cancelamento de uma pré-autorização criada</p> <p>Retorna o resultado do cancelamento da pré-autorização</p>

<u>PlugPagTransactionResult</u>	doEffectuatePreAuto(paymentData: PlugPagEffectuatePreAutoData) Realiza a efetivação e uma pré-autorização Retorna o resultado da efetivação da pré-autorização
<u>PlugPagTransactionResult</u>	doPreAutoCreate(plugPagPreAutoData: PlugPagPreAutoData) Realiza a criação de uma pré-autorização Retorna o resultado da criação de uma pré-autorização
<u>PlugPagNFCInfosResultDirectly</u>	detectNfcCardDirectly(cardType: Int, timeout: Int) Realiza a detecção NFC diretamente com o hardware sem nenhum tipo de gerenciamento extra Retorna o resultado da detecção do cartão NFC
<u>Int</u>	detectNfcRemoveDirectly(detectRemove: PlugPagNFCDetectRemoveCard) Realiza a detecção da remoção do cartão NFC diretamente com o hardware sem nenhum tipo de gerenciamento extra Retorna o resultado da remoção do cartão
<u>Int</u>	justAuthNfcDirectly(plugPagNFCAuthDirectly: PlugPagNFCAuthDirectly) Realiza a autenticação do cartão NFC diretamente com o hardware sem nenhum tipo de gerenciamento extra Retorna o resultado da autenticação do cartão

PlugPagCmdExchangeResult apduCommand(command: ByteArray, dataLength: Int)

Realiza a chamada APDU com os parametros de commando
a ser enviado

Retornar as informações do comando apdu enviado

PlugPagAbortResult

Essa classe contém dados resultantes de uma solicitação de cancelamento de operação.

Construtores

PlugPagAbortResult(int result)

Cria um container de dados resultantes de um cancelamento de operação com o código `result`.

Métodos

int getResult()

Retorna o código de resultado da solicitação de cancelamento de operação.

PlugPagActivationData

Essa classe contém dados para realizar uma ativação.

Construtores

PlugPagActivationData(String activationCode)

Cria um conjunto de informações necessários para iniciar uma ativação.

PlugPagApplIdentification

Essa classe representa a identificação de um aplicativo.

Construtores

`PlugPagApplIdentification(String name, String version)`

Cria uma identificação do aplicativo, definindo seu nome e sua versão com os valores de `name` e `version`, respectivamente.

Gera uma exceção se `name` ou `version` forem nulos ou vazios.

Os tamanhos limite para `name` e `version` são, respectivamente, 25 e 10 caracteres.

Métodos

`String` `getName()`

Retorna o nome do aplicativo.

`String` `getVersion()`

Retorna a versão do aplicativo.

PlugPagBeepData

Essa classe representa os dados de um beep.

Constantes

byte	FREQUENCY_LEVEL_0
	Código utilizado para frequência 0.
	As frequências variam de 0 (grave) a 6 (agudo).
	Valor: 0

byte	FREQUENCY_LEVEL_1
	Código utilizado para frequência 1.
	As frequências variam de 0 (grave) a 6 (agudo).
	Valor: 1

byte	FREQUENCY_LEVEL_2
	Código utilizado para frequência 2.
	As frequências variam de 0 (grave) a 6 (agudo).
	Valor: 2

byte	FREQUENCY_LEVEL_3
	Código utilizado para frequência 3.
	As frequências variam de 0 (grave) a 6 (agudo).
	Valor: 3

byte	FREQUENCY_LEVEL_4
	Código utilizado para frequência 4.
	As frequências variam de 0 (grave) a 6 (agudo).
	Valor: 4

byte FREQUENCE_LEVEL_5

Código utilizado para frequência 5.

As frequências variam de 0 (grave) a 6 (agudo).

Valor: 5

byte FREQUENCE_LEVEL_6

Código utilizado para frequência 5.

As frequências variam de 0 (grave) a 6 (agudo).

Valor: 6

Construtores

PlugPagBeepData(byte frequency, int duration)

Cria um conjunto de informações necessários para executar um beep.

Gera uma exceção se `frequency` não for uma das constantes.

Gera uma exceção se `duration` for menor que 0.

PlugPagException

Essa classe representa uma exceção PlugPag.

Construtores

PlugPagException()

Cria uma exceção básica.

`PlugPagException(String message)`

Cria uma exceção com uma mensagem customizada.

`PlugPagException(Throwable cause)`

Cria uma exceção com um stack trace.

`PlugPagException(String message, String errorCode)`

Cria uma exceção com uma mensagem e um código de erro customizados.

`PlugPagException(String message, Throwable cause, String errorCode)`

Cria uma exceção com uma mensagem, um stack trace e um código de erro customizados.

[PlugPagInitializationResult](#)

Essa classe contém os dados de um resultado de uma ativação ou desativação.

Construtores

`PlugPagInitializationResult(int result, String errorCode, String errorMessage)`

Cria um objeto para armazenar um conjunto de informações resultantes de uma ativação ou desativação.

[PlugPagEventData](#)

Essa classe representa um evento gerado pela biblioteca `PlugPag` para o aplicativo de integração.

Constantes

int	EVENT_CODE_CUSTOM_MESSAGE	Código de evento indicando mensagem customizada pela PlugPag. Valor: -2
int	EVENT_CODE_DEFAULT	Código padrão de evento. Utilizado quando nenhum evento foi enviado. Valor: -1
int	EVENT_CODE_WAITING_CARD	Código de evento indicando que o leitor está aguardando o usuário inserir o cartão. Valor: 0
int	EVENT_CODE_INSERTED_CARD	Código de evento indicando que o cartão foi inserido. Valor: 1
int	EVENT_CODE_PIN_REQUESTED	Código de evento indicando que o leitor está aguardando o usuário digitar a senha. Valor: 2
int	EVENT_CODE_PIN_OK	Código de evento indicando que a senha digitada foi validada com sucesso. Valor: 3

int	EVENT_CODE_SALE_END	Código de evento indicando o fim da transação. Valor: 4
int	EVENT_CODE_AUTHORIZING	Código de evento indicando que o terminal está aguardando autorização da senha digitada para prosseguir com a transação. Valor: 5
int	EVENT_CODE_INSERTED_KEY	Código de evento indicando que a senha foi digitada. Valor: 6
int	EVENT_CODE_WAITING_REMOVE_CARD	Código de evento indicando que o terminal está aguardando o usuário remover o cartão. Valor: 7
int	EVENT_CODE_REMOVED_CARD	Código de evento indicando que o cartão foi removido do terminal. Valor: 8
int	EVENT_CODE_CVV_REQUESTED	Código de evento indicando que foi solicitado o CVV. Valor: 9

int EVENT_CODE_CVV_OK

Código de evento indicando que o CVV foi inserido corretamente.

Valor: 10

int EVENT_CODE_CAR_BIN_REQUESTED

Código de evento indicando que foi solicitado o BIN.

Valor: 11

int EVENT_CODE_CAR_BIN_OK

Código de evento indicando que o BIN foi inserido corretamente.

Valor: 12

int EVENT_CODE_CAR HOLDER_REQUESTED

Código de evento indicando que foi solicitado o CVV.

Valor: 13

int EVENT_CODE_CAR HOLDER_OK

Código de evento indicando que o HOLDER foi inserido corretamente.

Valor: 14

int EVENT_CODE_ACTIVATION_SUCCESS

Código de evento indicando que a ativação foi feita corretamente.

Valor: 15

int EVENT_CODE_DIGIT_PASSWORD

Código de evento indicando que a um número da senha foi digitado.

Valor: 16

int EVENT_CODE_NO_PASSWORD

Código de evento indicando que a senha foi apagada.

Valor: 17

int EVENT_CODE_SALE_APPROVED

Código de evento indicando a venda foi aprovada.

Valor: 18

int EVENT_CODE_SALE_NOT_APPROVED

Código de evento indicando que a venda não foi aprovada.

Valor: 19

Construtores

PlugPagEventData(int eventCode)

Cria um identificador de evento gerado pela biblioteca para o aplicativo de integração, com o código `eventCode`.

Métodos

int getEventCode()

Retorna o código do evento gerado.

Métodos estáticos

String getDefaultMessage(int eventCode)

Retorna uma mensagem padrão para um dado código de evento.

Se o código de evento for inválido, retorna uma mensagem padrão.

PlugPagLedData

Essa classe representa os dados de um LED.

Constantes

byte	LED_BLUE	Código utilizado para acender o LED azul. Valor: 8
byte	LED_YELLOW	Código utilizado para acender o LED amarelo. Valor: 4
byte	LED_GREEN	Código utilizado para acender o LED verde. Valor: 2
byte	LED_RED	Código utilizado para acender o LED vermelho. Valor: 1
byte	LED_OFF	Código utilizado para desligar os LEDs. Valor: 0

Construtores

`PlugPagLedData (byte led)`

Define o padrão de LEDs a serem acesos.

Gera uma exceção se `led` não for uma das constantes.

PlugPagNFCAuth

Essa classe representa os dados para realizar uma autenticação NFC.

Construtores

`PlugPagNFCAuth (int type, byte blockNumber, byte[] password)`

Cria um conjunto de informações necessários para autenticar NFC.

O parâmetro `type` deve ser uma das seguintes constantes da classe `PlugPagNearFieldCardData`: `ISO14443_AB`, `EMV_AB`, `ONLY_A`, ***ONLY_B*** ou `ONLY_M`.

Caso não seja necessária uma senha para autenticar, o parâmetro `password` deve ser preenchido com valores `0xFF`.

Construtores

```
PlugPagNFCAuth (int type, byte blockNumber, byte[] password, em1KeyType: EM1KeyType)
```

Cria um conjunto de informações necessários para autenticar NFC.

O parâmetro `type` deve ser uma das seguintes constantes da classe `PlugPagNearFieldCardData`: `ISO14443_AB`, `EMV_AB`, `ONLY_A`, ***ONLY_B*** ou `ONLY_M`.

Caso não seja necessária uma senha para autenticar, o parâmetro `password` deve ser preenchido com valores `0xFF`.

Também é possível passar o tipo de autenticação através do atributo `em1KeyType` sendo default a autenticação do tipo A

PlugPagNearFieldCardData

Essa classe representa os dados de cartões NFC

Constantes

int	ISO14443_AB
	Código utilizado para o tipo ISO14443_AB.
	Valor: 0

int EMV_AB

Código utilizado para o tipo EMV_AB.

Valor: 1

int ONLY_A

Código utilizado para o tipo ONLY_A

Valor: 2

int ONLY_B

Código utilizado para o tipo ONLY_B.

Valor: 3

int ONLY_M

Código utilizado para o tipo ONLY_M.

Valor: 4

String NFC_PWD

Código utilizado para o tipo NFC_PWD.

Valor: pwd

String NFC_DATA

Código utilizado para o tipo NFC_DATA.

Valor: data

PlugPagSimpleNFCDData

Essa classe representa os dados de cartões NFC.

Construtores

```
PlugPagSimpleNFCDData(int cardType, int slot, byte[] value)
```

Cria um container de dados de um cartão NFC.

PlugPagNFCInfosResult

Essa classe contém dados do resultado das informações de um cartão NFC.

Construtores

```
PlugPagNFCInfosResult(int result, byte cardType, byte cid, byte[] other, Serializable  
serialNumber)
```

Cria um container de dados resultantes de uma operação para obter informações de um cartão NFC.

PlugPagPaymentData

Essa classe representa os dados de um pagamento.

É nessa classe que são definidas informações de tipo de pagamento, valor a ser pago e parcelas, além e outras informações gerenciais.

Construtores

```
PlugPagPaymentData(int paymentType, int amount, int installmentType, int installments,  
String userReference)
```

Cria um conjunto de informações necessários para iniciar um pagamento. O pagamento configurado será do tipo `paymentType`, com o valor `amount`, com parcelamento do tipo `installmentType`, com `installments` número de parcelas, identificado por `userReference`.

O parâmetro `amount` definido é o valor em centavos a ser pago. Para um pagamento de R\$ 1,50, o `amount` deverá ser de 150.

O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres.

Gera uma exceção se o `userReference` for nulo ou vazio.

`PlugPagPaymentData(int paymentType, int amount, int installmentType, int installments, String userReference, Boolean printReceipt)`

Cria um conjunto de informações necessários para iniciar um pagamento. O pagamento configurado será do tipo `paymentType`, com o valor `amount`, com parcelamento do tipo `installmentType`, com `installments` número de parcelas, identificado por `userReference`.

O parâmetro `amount` definido é o valor em centavos a ser pago. Para um pagamento de R\$ 1,50, o `amount` deverá ser de 150.

O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres.

O parâmetro `printReceipt` indicará se deverá ser impresso os comprovantes da transação.

Gera uma exceção se o `userReference` for nulo ou vazio.

Métodos

int	<code>getAmount()</code>
	Retorna o valor a ser pago, em centavos.

int	<code>getInstallments()</code>
	Retorna o número de parcelas do pagamento.

int	<code>getInstallmentType()</code>
	Retorna o tipo de parcelamento.
	Valores: <code>PlugPag.INSTALLMENT_TYPE_A_VISTA</code> ou <code>PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR</code>

int `getType()`

Retorna o tipo de pagamento.

Valores: `PlugPag.TYPE_CREDITO`, `PlugPag.TYPE_DEBITO`,
`PlugPag.TYPE_VOUCHER`, `PlugPag.TYPE_QRCODE` ou
`PlugPag.TYPE_QRCODE_CREDITO`

String `getUserReference()`

Retorna o código de venda.

[PlugPagPaymentData.Builder](#)

Construtor de objetos `PlugPagPaymentData`.

Construtores

`Builder()`

Cria um construtor de objetos `PlugPagPaymentData`.

Métodos

`PlugPagPaymentData build()`

Cria um `PlugPagPaymentData` com os dados armazenados no
`Builder`.

Builder `setAmount(int amount)`

Define o valor a ser pago.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

O valor de `amount` deve ser fornecido em centavos. Por exemplo, se o valor desejado é de R\$1,50, deve-se passar o valor `150`.

Gera uma exceção se `amount` não for maior do que zero.

Builder `setInstallments(int installments)`

Retorna a quantidade de parcelas do pagamento.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Se `installments` for igual a `1`, o tipo de parcelamento é automaticamente definido para `PlugPag.INSTALLMENT_TYPE_A_VISTA`.

Gera uma exceção se `installments` não for maior do que zero.

Builder `setInstallmentType(int installmentType)`

Define o tipo de parcelamento.

Valores válidos para `installmentType` são `PlugPag.INSTALLMENT_TYPE_A_VISTA` e `PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR`.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Gera uma exceção se `installmentType` for inválido.

Builder

setType(int type)

Define o tipo de pagamento.

Valores válidos para `type` são `PlugPag.TYPE_CREDITO`, `PlugPag.TYPE_DEBITO`, `PlugPag.TYPE_VOUCHER`, `PlugPag.TYPE_QRCODE` e `PlugPag.TYPE_QRCODE_CREDITO`.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Gera uma exceção se `type` for inválido.

Builder

setUserReference(String userReference)

Define o código de venda.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

O valor de `userReference` deve conter apenas letras (não acentuadas) e números. Esse campo é limitado a 10 caracteres.

Gera uma exceção se `userReference` for nulo ou vazio.

PlugPagTransactionResult

Essa classe representa o resultado de uma transação.

Construtores

```
PlugPagTransactionResult (String message, String errorCode, String transactionCode, String transactionId, String date, String time, String hostNsu, String cardBrand, String bin, String holder, String userReference, String terminalSerialNumber, String amount, String availableBalance, String cardApplication, String label, String holderName, String extendedHolderName, Int result, String readerModel, String nsu, String autoCode, Char installments, Int originalAmount, String buyerName, Int paymentType, String typeTransaction, String appIdentification, String cardHash, String mPreAutoDueDate, String mPreAutoOriginalAmount)
```

Cria um objeto para armazenar um conjunto de informações resultantes de uma transação.

```
PlugPagTransactionResult(String message, String errorCode ,String transactionCode, String transactionId, String date, String time, String hostNsu, String cardBrand, String bin, String holder, String userReference, String terminalSerialNumber, String amount, String availableBalance, String cardApplication, String label, String holderName, String extendedHolderName, String readerModel, String nsu, String autoCode, Char installments, Int originalAmount, String buyerName, Int paymentType, String typeTransaction, String appIdentification, String cardHash, String mPreAutoDueDate, String mPreAutoOriginalAmount)
```

Cria um objeto para armazenar um conjunto de informações resultantes de uma transação, adicionando o código de resultado `result`.

Métodos

String getAmount()

Retorna o valor transacionado.

String getAvailableBalance()

Retorna o saldo da conta, caso o método de pagamento seja `PlugPag.TYPE_VOUCHER`.

String getBin()

Retorna os 4 (quatro) últimos dígitos do cartão utilizado.

String getCardApplication()

Retorna a aplicação do cartão.

String getCardBrand()

Retorna a bandeira do cartão utilizado.

String getBuyerName()

Retorna o nome do proprietário da carteira PIX

String getDate()

Retorna a data da transação.

String getErrorCode()

Se um erro ocorreu durante a transação, retorna o código de erro.

String getExtendedHolderName()

Retorna o nome completo do titular do cartão utilizado.

String getHolder()

Retorna os 4 últimos dígitos do cartão utilizado.

String getHolderName()

Retorna o nome do titular do cartão utilizado.

String getHostNsu()

Retorna um identificador único do host (servidor).

String getNsu()

Retorna o NSU da transação

String getLabel()

Retorna o label do cartão utilizado.

String getMessage()

Retorna uma mensagem do resultado da transação, definida pela biblioteca.

int getResult()

Retorna o código do resultado.

String getTerminalSerialNumber()

Retorna o número de série do terminal ou leitor utilizado para efetuar o pagamento.

String getTime()

Retorna o horário da transação.

String getTransactionCode()

Retorna o código da transação.

String	getTransactionId()	Retorna o ID da transação.
String	getUserReference()	Retorna o código de venda o pagamento efetuado.
String	getOriginalAmount()	Retorna o código de venda o pagamento efetuado.
String	getInstallments()	Retorna a quantidade de parcelas da transação
String	getReaderModel()	Retorna o modelo do equipamento
String	getTypeTransaction()	Retorna o tipo da transação
String	getPaymentType()	Retorna qual o tipo de pagamento a transação se relaciona
String	getAppIdentification()	Retorna qual o valor preenchido no parametro appNamenote
String	getAutoCode()	Retorna o codigo de autorização
String	getPreAutoDueDate()	Retorna a Data do vencimento da Pré-Autorização criada

String getPreAutoOriginalAmont()

Retorna o Valor do saldo original pré autorizado.

PlugPagTransactionResult.Builder

Construtor de objetos `PlugPagTransactionResult`.

Construtores

Builder()

Cria um construtor de objetos `PlugPagTransactionResult`.

Métodos

`PlugPagTransactionResult` `build()`

Constrói uma instância da classe `PlugPagTransactionResult` utilizando os dados armazenados.

Builder

`setAmount(String amount)`

Define o valor da transação.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder

`setAvailableBalance(String availableBalance)`

Define o saldo disponível.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder

`setBin(String bin)`

Define o BIN do cartão utilizado na transação.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Builder	<p><code>setCardApplication(String cardApplication)</code></p> <p>Define a aplicação do cartão utilizado na transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setCardBrand(String cardBrand)</code></p> <p>Define a bandeira do cartão utilizado na transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setDate(String date)</code></p> <p>Define a data da transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setExtendedHolderName(String extendedHolderName)</code></p> <p>Define o nome completo do titular do cartão utilizado na transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setHolder(String holder)</code></p> <p>Define o nome do titular do cartão.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>

Builder	<p><code>setHolderName(String holderName)</code></p> <p>Define o nome do titular do cartão.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setHostNsu(String hostNsu)</code></p> <p>Define o NSU do host que executou a transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setLabel(String label)</code></p> <p>Define o label do cartão utilizado.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setMessage(String message)</code></p> <p>Define a mensagem do resultado da transação que será construído.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setTerminalSerialNumber(String terminalSerialNumber)</code></p> <p>Define o número de série do terminal ou leitor utilizado na transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>

Builder	<p><code>setTime(String time)</code></p> <p>Define o horário da transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setTransactionCode(String transactionCode)</code></p> <p>Define o código da transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setTransactionId(String transactionId)</code></p> <p>Define o ID da transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>
Builder	<p><code>setUserReference(String userReference)</code></p> <p>Define o código de venda da transação.</p> <p>Retorna a referência do próprio <code>Builder</code> para chamadas encadeadas.</p>

PlugPagVoidData

Essa classe representa os dados de um estorno.

É nessa classe que são definidos dados necessários para solicitar o estorno de um pagamento.

Construtores

PlugPagVoidData(String transactionCode, String transactionId, Boolean printReceipt, Int VoidType)

Cria um conjunto de informações para solicitar o estorno de um pagamento identificado pelo `transactionCode` e `transactionId` fornecidos.

O parâmetro `printReceipt` é opcional e indicará se deverá ser impresso os comprovantes da transação.

O parâmetro `voidType` é opcional e indicará qual o tipo de estorno ira ser realizado.

Gera uma exceção se `transactionCode` for nulo ou vazio.

PlugPagVoidData(String transactionCode, String transactionId, Int VoidType)

Cria um conjunto de informações para solicitar o estorno de um pagamento identificado pelo `transactionCode` e `transactionId` fornecidos.

O parâmetro `voidType` é opcional e indicará qual o tipo de estorno ira ser realizado.

Gera uma exceção se `transactionCode` for nulo ou vazio.

PlugPagVoidData(String transactionCode, String transactionId)

Cria um conjunto de informações para solicitar o estorno de um pagamento identificado pelo `transactionCode` e `transactionId` fornecidos.

Gera uma exceção se `transactionCode` for nulo ou vazio.

Métodos

String	getTransactionCode()
--------	----------------------

Retorna o código da transação que será estornada.

String	getTransactionId()
--------	--------------------

Retorna o ID da transação que será estornada.

Int	getPrintReceipt()
-----	-------------------

Retorna se será impresso comprovante na transação.

Int	getVoidType()
-----	---------------

Retorna o tipo de estorno que será realizado.

PlugPagVoidData.Builder

Construtor de objetos `PlugPagVoidData`.

Construtores

Builder()

Cria um construtor de objetos `PlugPagVoidData`.

Métodos

PlugPagVoidData build()

Constrói uma instância da classe `PlugPagVoidData` utilizando os dados armazenados.

Builder setTransactionCode(String transactionCode)

Define o código da transação.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

Gera uma exceção se `transactionCode` for nulo ou vazio.

Builder setTransactionId(String transactionid)

Define o ID da transação.

Retorna a referência do próprio `Builder` para chamadas encadeadas.

PlugPagCustomPrinterLayout

Essa classe representa os elementos a serem customizados da dialog de impressão da via do cliente.

Construtor

`PlugPagCustomPrinterLayout()`

Cria um construtor de objetos `PlugPagCustomPrinterLayout`.

Métodos

void	<code>setButtonBackgroundColor(String hexaCodeColor)</code> Modifica a cor de fundo dos botões da dialog.
void	<code>setButtonBackgroundColorDisabled(String hexaCodeColor)</code> Modifica a cor de fundo dos botões da dialog quando desativados.
void	<code>setCancelTextColor(String hexaCodeColor)</code> Altera a cor do texto do botão de cancelar da dialog.
void	<code>setSendSMSTextColor(String hexaCodeColor)</code> Altera a cor do texto do botão de enviar sms da dialog.
void	<code>setConfirmTextColor(String hexaCodeColor)</code> Altera a cor do texto do botão de confirmar da dialog.
void	<code>setTitle(String titleText)</code> Seta o texto a ser mostrado na dialog.

void	setTitleColor(String hexaCodeColor)	Altera a cor do texto mostrado na dialog.
void	setWindowBackgroundColor(String hexaCodeColor)	Modifica a cor de fundo da dialog.
void	setMaxTimeShowPopup(Integer time)	Altera o tempo de exibição do popup. Se existir valor para esse atributo o popup fecha automaticamente de acordo com o tempo estabelecido em segundos.

Todos os itens são opcionais. Caso não sejam setados, obterão seus valores defaults.

PlugPagNFCResult

Essa classe representa o retorno de uma leitura ou escrita a um cartão NFC.

Construtor

PlugPagNFCResult(int startSlot, int endSlot, HashMap<String, Byte[]>[] slots, int result)

Métodos

int	getStartSlot()	Retorna o primeiro slot a ser escrito/lido
int	getEndSlot()	Retorna o último slot a ser escrito/lido.

HashMap<String, Byte[]>[] getSlots()

Retorna as informações as serem escritas/lidas de cada slot.

No total, são 64 slots onde, cada slot, possui um HashMap contendo duas informações: data e pwd. Pwd retorna a senha de deste slot e data retorna o valor que foi lido/escrito naquele slot.

int getResult()

Retorna 1 para sucesso e -1 para falha.

PlugPagPrintResult

Essa classe representa o retorno de uma requisição de impressão pelo PlugPagService.

Construtor

PlugPagPrintResult(int result, String message, String errorCode)

Métodos

int	getResult()	Retorna PlugPag. RET_OK quando sucesso.
int	getMessage()	Retorna a mensagem de erro da operação.
String	getErrorCode()	Retorna o código de erro da operação.

PlugPagPrinterData

Essa classe representa os dados de uma impressão a ser realizada.

Construtor

PlugPagPrinterData(String filePath, Int printerQuality, Int step)

Métodos

String filePath	getFilePath() Retorna o caminho de arquivo a ser impresso.
int printerQuality	getPrinterQuality() Retorna a qualidade da impressão. Os valores podem variar de 1 a 4, onde 4 indica a maior qualidade da impressão.
int step	getStep() Retorna o espaçamento a ser feito após a impressão terminar.

PlugPagMockResult

Essa classe contém os atributos escolhidos para algum caso de mock

Construtor

PlugPagMockResult(val codeResult: Int, val errorCode: String)

Constantes

int	SUCCESS_CODE	
	Código utilizado para indicar que o retorno do mock será sucesso	
	Entrada: 0	
	Saída:	
	Código	resultado: 0
	Código do	erro: 0
	Mensagem: "Sucesso"	

```
int ERROR_TIMEOUT_CODE
```

Código utilizado para definir que o retorno do mock será um erro de timeout

Entrada: 1

Saída:

Código resultado: -1004

Código do erro: A11

Mensagem: "Operadora de telefonia indisponível. Tente novamente."

```
int ERROR_TRANSACTION_CODE
```

Código utilizado para definir que o retorno do mock será um erro de transação

Entrada: 2

Saída:

Código resultado: -1004

Código do erro: A012

Mensagem: "Erro de SIMCARD (chip) Contate a central de suporte."

```
int ERROR_SIM_CARD_CODE
```

Código utilizado para definir que o resultado de mock será um erro de cartão

SIM

Entrada: 3

Saída:

Código resultado: -1004

Código do erro: A15

Mensagem: "Falha na transacao. Tente novamente."

int

Código utilizado para definir que o caso de mock será um erro de comunicação

Entrada: 4

Saída:

Código resultado: -1004

Código do erro: A104

Mensagem: "Problema na comunicacao Contate o PagSeguro."

int

Código utilizado para definir que o caso de mock será um erro de função indisponível

Entrada: 5

Saída:

Código resultado: -1004

Código do erro: S510

Mensagem: "Funcao indisponivel no momento. Cancele pelo site PagSeguro."

int

Código utilizado para definir que o caso de mock será serial não identificado ou código de ativação inválido

Entrada: 6

Saída:

Código resultado: -1004

Código do erro: M5004

Mensagem: "Numero serial do leitor nao identificado."

int ERROR_TRANSACTION_REFUNDED_BEFORE

Código utilizado para definir que o caso de mock será um erro de que uma transação já foi estornada anteriormente.

Entrada: 7

Saída:

Código	resultado:	-1004
Código	do	erro: S16
Mensagem: "Transacao ja estornada."		

int ERROR_REFUND_DATE_EXCEED

Código utilizado para definir que a resposta do mock será que a data limite para o estorno foi ultrapassada.

Entrada: 8

Saída:

Código	resultado:	-1004
Código	do	erro: M3005
Mensagem: "Data maxima para estorno da transacao foi excedida."		

int ERROR_INVALID MOCK_SITUATION

Código utilizado para indicar que a resposta do mock selecionado não está disponível para determinada operação.

Entrada: 9

Saída:

Código	resultado:	-8000
Código	do	erro: MK800
Mensagem: "Codigo do mock e invalido para esta operacao."		

int	CÓDIGOS NÃO MAPEADOS
	Código utilizado para indicar que foi solicitado um código de mock não mapeado.
	Entrada: valor < 0 ou valor > 9
	Saída:
	Código resultado: -1999
	Código do erro: -1
	Mensagem: "Codigo do Mock desconhecido."

PreferredNetwork

Essa classe contém as constantes de preferências de conexão.

Constantes

int	NETWORK_ALL
	Código utilizado para todos os tipos de conexões.
	Valor: 1

int	NETWORK_3G
	Código utilizado para conexão 3G
	Valor: 2

int	NETWORK_2G
	Código utilizado para conexão 2G
	Valor: 3

TerminalCapabilities

Essa classe contém as constantes com as funcionalidades do terminal.

Constantes

int	MODULE_MAG	Código utilizado para modulo de Leitor tarja	Valor: 1
int	MODULE_ICC	Código utilizado para modulo de Leitor de chip	Valor: 2
int	MODULE_PICC	Código utilizado para modulo de ContacLess	Valor: 3
int	MODULE_PED	Código utilizado para modulo de Teclado seguro	Valor: 4
int	MODULE_KEYBOARD	Código utilizado para modulo de Teclado externo	Valor: 5

int MODULE_PRINTER

Código utilizado para modulo de impressora

Valor: 6

int MODULE_BT

Código utilizado para modulo de bluetooth

Valor: 7

int MODULE_CASH_BOX

Código utilizado para modulo de Caixa registradora

Valor: 8

int MODULE_CUSTOMER_DISPLAY

Código utilizado para modulo de Tela auxiliar

Valor: 9

int MODULE_ETHERNET

Código utilizado para modulo de Rede

Valor: 10

int MODULE_FINGERPRINT_READER

Código utilizado para modulo de Leitor de digital

Valor: 11

int MODULE_G_SENSOR

Código utilizado para modulo de acelerômetro

Valor: 12

int	MODULE_HDMI
-----	-------------

Código utilizado para modulo de Saída hdmi

Valor: 13

PlugPagPrinterListener

Essa classe contém os métodos chamados no sucesso ou falha de uma impressão.

Métodos

void	onError(PlugPagPrintResult data)
------	----------------------------------

Callback com as informações da impressão no caso de falha.

void	onSuccess(PlugPagPrintResult data)
------	------------------------------------

Callback com as informações da impressão no caso de sucesso.

PlugPagEventListener

Essa classe contém o método chamado quando existem novos eventos a serem informados ao integrador sobre o processo de pagamento, estorno, desativação ou ativação.

Métodos

void	onEvent(PlugPagEventData data) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.
------	---

PlugPagPaymentListener

Essa classe contém os métodos chamados durante o processo de pagamento assíncrono.

Métodos

void	onError(PlugPagTransactionResult result) Callback com as informações da pagamento no caso de sucesso.
void	onSuccess(PlugPagTransactionResult result) Callback com as informações da pagamento no caso de sucesso.
void	onPaymentProgress(PlugPagEventData eventData) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.

void	onPrinterSuccess(PlugPagPrintResult printerResult)
	Callback com as informações da impressão no caso de sucesso.

void	onPrinterError(PlugPagPrintResult printerResult)
	Callback com as informações da impressão no caso de falha.

void	onPrinterError(PlugPagPrintResult printerResult)
------	--

PlugPagNFCListener

Essa classe contém os métodos chamados no sucesso ou falha de uma escrita/leitura NFC assíncrona.

Métodos

void	onSuccess(PlugPagNFCResult plugPagNFCResult)
	Callback com as informações da leitura/escrita NFC no caso de sucesso.

void	onError(String errorMessage)
	Callback com a mensagem de erro da leitura/escrita NFC.

PlugPagLastTransactionListener

Essa classe contém os métodos chamados na requisição assíncrona de última transação aprovada.

Métodos

void	onRequestedLastTransaction(PlugPagTransactionResult result) Callback com as informações da última transação realizada, seja pagamento ou estorno.
void	onError(String errorMessage) Callback com a mensagem de erro da busca pela última transação realizada.

PlugPagIsActivatedListener

Essa classe contém os métodos chamados na verificação assíncrona de status do terminal.

Métodos

void	onIsActivated(Boolean isActivated) Callback com a informação se o terminal está ou não ativado.
void	onError(String errorMessage) Callback com a mensagem de erro ao buscar status do terminal.

PlugPagInstallmentsListener

Essa classe contém os métodos chamados no cálculo de parcelas.

Métodos

void	onCalculateInstallments(Array<String> installments) Callback com informações de cada parcela, seguindo o padrão: "R\$ ##,##".
void	onError(String errorMessage) Callback com a mensagem de erro ao calcular parcelas.

PlugPagActivationListener

Essa classe contém os métodos chamados no processo assíncrono de ativação ou desativação do terminal.

Métodos

void	onActivationProgress(PlugPagEventData eventData) Callback que envia ao integrador informações importantes e ações a serem tomadas no processo de pagamento, estorno, desativação ou ativação.
void	onSuccess(PlugPagInitializationResult result) Callback com as informações da ativação/desativação no caso de sucesso.

void	onError(String errorMessage)
	Callback com as informações da ativação/desativação no caso de falha.

PlugPagAbortListener

Essa classe contém os métodos chamados no processo assíncrono de abort.

Métodos

void	onAbortRequested(Boolean abortRequested)
	Callback com informação se o abort foi solicitado com sucesso.
void	onError(String errorMessage)
	Callback com as informações do erro na requisição de abort.

PlugPagCardInfoResult

Essa classe contém os dados do cartão, retornados pelo método GetCardData().

Atributos

result	Código de erro, quando houver, zero quando não houver.
message	Mensagem de erro, quando houver, nullo quando não houver.
bin	Bin do cartão lido.
holder	Holder do cartão lido.
cardHolder	Card holder mascarado do cartão lido.

PlugPagPreAutoData

A criação da pré-autorização é semelhante ao fluxo de venda crédito comum e consiste quando o usuário reserva um determinado valor no cartão de crédito associado a transação para cobrança futura. Depois de criada, a pré-autorização está apta a ser cancelada ou efetivada.

Para realizar a criação deve-se utilizar o método do `doPreAutoCreate`, que por sua vez recebe como parâmetro um objeto do tipo `PlugPagPreAutoData` que contém seguintes membros a serem preenchidos de maneira **obrigatória**:

Atributos

Int	amount	Define o valor da transação.
String	userReference	Define o código de venda da transação.
String	printReceipt	Define se deseja imprimir o comprovante
Int	installmentType	Define o tipo de parcelamento. Valores válidos para <code>installmentType</code> da pré-autorização são <code>PlugPag.INSTALLMENT_TYPE_A_VISTA</code> e <code>PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR</code> .
Int	installments	Define a quantidade de parcelas

PlugPagEffectuatePreAutoData

A efetivação consiste em confirmar uma pré-autorização existente, onde o valor desta deve ser menor ou igual ao valor especificado na criação. O método da lib `doEffectuatePreAuto` deve ser utilizado para efetivar uma pré-autorização e este recebe como parâmetro um objeto do tipo `PlugPagEffectuatePreAutoData`, que possui os seguintes campos que devem ser preenchidos de maneira **obrigatória**:

Atributos

Int	amount	Define o valor da transação.
String	userReference	Define o código de venda da transação.
String	printReceipt	Define se deseja imprimir o comprovante
String	transactionId	Define o ID da transação.
String	transactionCode	Define o código da transação.

PlugPagNFCInfosResultDirectly

Essa classe contém dados do resultado das informações de um cartão NFC.

Construtores

```
PlugPagNFCInfosResultDirectly(int result, byte cardType, byte cid, byte[] other, byte[] serialNumber)
```

Cria um container de dados resultantes de uma operação para obter informações de um cartão NFC através do método `detectNfcCardDirectly`

Atributos

result	Código de erro, quando houver, zero quando não houver.
cardType	Tipo de cartão para ser autenticado Passar o parâmetro <code>PlugPagNFCInfosResultDirectly</code> e selecionar o tipo desejado
cid	Código identificador do cartão
other	Informações do cartão
serialNumber	Serial number do cartão

PlugPagNFCDetectRemoveCard

Essa classe contém atributos necessários para conseguir realizar a chamada de detecção de remoção de um cartão NFC

Construtores

`PlugPagNFCDetectRemoveCard(detectRemoveCard:PlugPagNearFieldRemoveCardData, cardId: Byte)`

Cria um container de dados para de uma operação para obter informações de um cartão NFC através do método `detectNfcRemoveDirectly`

Atributos

<code>detectRemoveCardType</code>	Enum com o tipo de remoção à ser enviado para a lib: HALT, REMOVE e ENV
-----------------------------------	--

<code>cardId</code>	Código identificador do cartão
---------------------	--------------------------------

PlugPagNearFieldRemoveCardData

Enum que contém os tipos permitidos para serem passados para a biblioteca para realizar a chamada do método de remoção de cartão NFC.

Modos

HALT	Sai depois de enviar o comando de remoção para o cartão
REMOVE	Identifica a intenção de observar quando o cartão é removido para o hardware
EMV	Cumpri com o modo de remoção do cartão Padrão EMV Não Conectado

PlugPagNFCAuthDirectly

Essa classe representa os dados para realizar uma autenticação NFC dando a possibilidade de escolher o tipo de autenticação A ou B.

Construtores

```
PlugPagNFCAuthDirectly (slotNumber: Byte, password: ByteArray, em1KeyType: EM1KeyType, serialNumber: ByteArray)
```

Cria um conjunto de informações necessários para autenticar NFC diretamente com a lib através do método justAuthNfcDirectly.

Atributos

slotNumber	Identifica qual bloco NFC utilizar para a autenticação
password	Chave para a autenticação NFC, exemplo: byte[] key = { (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF };
em1KeyType	Identifica o tipo de autenticação NFC (tipo A ou Tipo B)
serialNumber	Serial number do cartão NFC

EM1KeyType

Esse Enum representa a estrutura de opções para a autenticação NFC, sendo telas tipo A ou Tipo B.

Modos

TYPE_A	Seleciona o modo tipo A para a autenticação
TYPE_B	Seleciona o modo tipo B para a autenticação

PlugPagCmdExchangeResult

Essa classe representa os dados de resposta para as chamadas APDU, onde trafega as informações binárias recebidas do cartão

Atributos

cmd	Seleciona o modo tipo A para a autenticação
-----	---

Exemplos

Seguem abaixo alguns exemplos de camadas dos métodos do **PlugPagService Wrapper** para realizar transações.

As formas de fazer as chamadas e de tratar os valores retornados vão depender da implementação do seu aplicativo, porém é importante que sua aplicação controle ações de duplo clique para prevenir que o serviço seja chamado duas vezes.

Pagamentos

Pagamento de R\$250,00, no crédito, à vista:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_CREDITO,  
            25000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Pagamento de R\$300, no crédito, parcelado em 3 parcelas:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_CREDITO,  
            30000,  
            PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR,  
            3,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Pagamento de R\$150,00, no débito:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_DEBITO,  
            15000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Pagamento de R\$50, no voucher:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_VOUCHER,  
            5000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Estornar um pagamento

```
public void voidPayment(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.voidPayment(  
            "transactionCode",  
            "transactionId");  
  
        // Trata o resultado do estorno  
        ...  
    }  
}
```

Verificar autenticação

```
public void checkAuthentication(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Verifica autenticação  
    boolean authenticated = plugpag.isAuthenticated();  
  
    if (authenticated) {  
        // Usuário autenticado  
        ...  
    } else {  
        // Usuário não autenticado  
        ...  
    }  
}
```

Invalidar autenticação

```
public void logout(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Invalida a autenticação existente  
    plugpag.invalidateAuthentication();  
    ...  
}
```

Solicitar ativação

```
public void requestAuth(Activity activity) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugPag = new PlugPag(activity, appIdentification);  
  
    // Cria objeto com informação do código de ativação  
    PlugPagActivationData plugPagActivationData =  
        new PlugPagActivationData("SeuCodigoDeAtivação");  
  
    // Solicita autenticação  
    PlugPagInitializationResult result =  
        plugPag.initializeAndActivatePinPad(plugPagActivationData);  
  
    if(result.getResult() == PlugPag.RET_OK) {  
        //Sucesso  
    } else {  
        //Erro  
    }  
    ...  
}
```

O resultado da autenticação será retornado através da classe `PlugPagInitializationResult`.

Se a autenticação for efetuada com sucesso, o método `getResult()` irá retornar valor igual a `PlugPag.RET_OK`. Caso contrário, mais informações do erro podem ser encontradas nos métodos `getErrorCode()` e `getErrorMessage()`.

Obter versão da biblioteca

```
public void getLibVersion(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Obtém a versão da biblioteca  
    String version = plugpag.getLibVersion();  
}
```

Reimpressão da via do estabelecimento

```
public void reprintStablishmentReceipt() {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Obtém a versão da biblioteca  
    PlugPagPrintResult result = plugpag.reprintStablishmentReceipt();  
}
```

Reimpressão da via do cliente

```
public void reprintCustomerReceipt() {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Obtém a versão da biblioteca  
    PlugPagPrintResult result = plugpag.reprintCustomerReceipt();  
}
```

Calcular parcelas

```
public void calculateInstallments() {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Obtém a versão da biblioteca  
    Array<String> installments = plugpag.calculateInstallments(saleValue);  
}
```

O resultado retornado será uma `String` contendo a quantidade de parcelas permitidas mais o valor de cada parcela, seguindo o padrão “<quantidadeDeParcelas>x R\$ <valorDaParcela>”.

Customizar dialog de impressão da via do cliente

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_DEBITO,  
            15000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Cria a customização da dialog de impressão da via do cliente  
    PlugPagCustomPrinterLayout customDialog = new PlugPagCustomPrinterLayout();  
    customDialog.setTitle("Imprimir via do client?");  
    customDialog.setButtonBackgroundColor("#00ff33");  
    customDialog.setConfirmText("Yes");  
    customDialog.setCancelText("No");  
  
    // Não obrigatório  
    customDialog.setMaxTimeShowPopup(5) // 5 segundos  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        plugPag.setPlugPagCustomPrinterLayout(customDialog);  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```


Ler cartão NFC

```
public void readNFCCard() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
    PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();  
    dataCard.setStartSlot(1);  
    dataCard.setEndSlot(1);  
  
    // Lê um cartão NFC  
    PlugPagNFCResult result = plugpag.readFromNFCCard(dataCard);  
}
```

O resultado da leitura será retornado através da classe `PlugPagNFCResult`.

Se a leitura for efetuada com sucesso, o método `getResult()` irá retornar valor igual a 1. Do contrário, o método `getResult()` retornará -1.

Escrever no cartão NFC

```
public void writeToNFCCard() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    String info = "teste_com16bytes";  
    byte[] infoBytes = info.getBytes();  
  
    PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();  
    dataCard.setStartSlot(1);  
    dataCard.setEndSlot(2);  
    dataCard.getSlots()[1].put("data", infoBytes);  
  
    // Escreve em um cartão NFC  
    int result = plugpag.writeToNFCCard(dataCard);  
}
```

O resultado da escrita será retornado através da classe `PlugPagNFCResult`.

Se a escrita for efetuada com sucesso, o método `getResult()` irá retornar valor igual a 1. Do contrário, o método `getResult()` retornará -1.

Abortar operação de leitura/escrita no cartão NFC

```
public void abortNFC() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    // Escreve em um cartão NFC  
    int result = plugpag.abort();  
}
```

O resultado do abort será retornado através da classe `PlugPagNFCResult`.

Se o abort for efetuada com sucesso, o método `getResult()` irá retornar valor igual a 1. Do contrário, o método `getResult()` retornará -1.

Ler cartão

```
public void readCard() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    // Lê cartão  
    PlugPagCardInfoResult dataCard = plugpag.getCardData();  
}
```

O resultado da leitura será retornado através da classe `PlugPagCardInfoResult`.

Imprimir arquivo

```
public void printFile() {
    // Cria a referência do PlugPag
    PlugPag plugPag = new PlugPag(context);

    // Cria objeto com informações da impressão
    PlugPagPrinterData data = new PlugPagPrinterData(
        Environment.getExternalStorageDirectory().getAbsolutePath() +
        "/SeuDiretorio/SeuArquivo",
        4,
        10 * 12));

    PlugPagPrinterListener listener = new PlugPagPrinterListener() {
        @Override
        public void onError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
            plugPagPrintResult.getMessage(); // Mensagem de erro
            plugPagPrintResult.getErrorCode(); // Código de erro
        }
    };

    //Seta listener de impressão
    plugPag.setPrinterListener(listener)

    // Imprime arquivo
    PlugPagPrintResult result = plugPag.printFromFile(plugPagPrinterData)

    if (result == PlugPag.RET_OK) {
        //sucesso
    }
}
```

Buscar última transação aprovada

```
public void getLastApprovedTransaction(Context context) {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    PlugPagTransactionResult lastApprovedTransaction =  
        plugpag.getLastApprovedTransaction();  
}
```

Pré-Autorização

Criar pré-autorização

```
public Observable<PlugPagTransactionResult> doPreAutoCreate(int value, int  
installmentType, int installments) {  
    return Observable.create(emitter -> {  
        PlugPagPreAutoData plugPagPreAutoData = new PlugPagPreAutoData(  
            value,  
            installmentType,  
            installments,  
            true  
            "userref"  
        );  
  
        PlugPagTransactionResult plugPagTransactionResult =  
mPlugPag.doPreAutoCreate(plugPagPreAutoData);  
        // Trata o retorno do PlugPagTransactionResult  
  
    });  
}
```

Efetivar pré-autorização

```
public Observable<PlugPagTransactionResult> doPreAutoEffectuate(  
    int value,  
    String transactionId,  
    String transactionCode  
) {  
    return Observable.create(emitter -> {  
        PlugPagEffectuatePreAutoData plugPagEffectuatePreAutoData = new  
PlugPagEffectuatePreAutoData(  
            value,  
            true,  
            transactionId,  
            transactionCode,  
            "userref"  
        );  
  
        PlugPagTransactionResult plugPagTransactionResult =  
mPlugPag.doEffectuatePreAuto(plugPagEffectuatePreAutoData);  
        // Trata o retorno do PlugPagTransactionResult  
    });  
}
```

Cancelar Pré-auto

```
public Observable<PlugPagTransactionResult> doPreAutoCancel(String transactionId,  
String transactionCode) {  
    return Observable.create(emitter -> {  
        PlugPagTransactionResult plugPagTransactionResult =  
mPlugPag.doPreAutoCancel(transactionId, transactionCode);  
  
        // Trata o retorno do PlugPagTransactionResult  
    });  
}
```

Buscar Pré-autorização

```
public Observable<PlugPagTransactionResult> getPreAutoData() {  
    return Observable.create(emitter -> {  
        PlugPagTransactionResult plugPagTransactionResult =  
mPlugPag.getPreAutoData();  
        // Trata o retorno do PlugPagTransactionResult  
    });  
}
```

NFC

detectNfcCardDirectly

```
public Observable<PlugPagNFCInfosResultDirectly> detectCardDirectly() {
    return Observable.create(emitter -> {
        try {
            // Starta a antena NFC
            int resultStartNfc = mPlugPag.startNFCCardDirectly();

            // Realiza a autenticação NFC
            PlugPagNFCInfosResultDirectly plugPagNFCInfosResult =
mPlugPag.detectNfcCardDirectly(PlugPagNearFieldCardData.ONLY_M, 20);

            if (plugPagNFCInfosResult.getResult() != RET_OK) {
                emitter.onError(new PlugPagException("Cartão não identificado"));
                emitter.onComplete();
                return;
            }

            emitter.onNext(plugPagNFCInfosResult);
            // Stopa a antena NFC
            mPlugPag.stopNFCCardDirectly();
        } catch (Exception e) {
            e.printStackTrace();
            emitter.onError(e);
        }

        emitter.onComplete();
    });
}
```

detectRemoveCardDirectly

```
public Observable<Integer> detectRemoveCardDirectly() {
    return Observable.create(emitter -> {
        try {
            // Starta a antena NFC
            int resultStartNfc = mPlugPag.startNFCCardDirectly();
            PlugPagNFCInfosResultDirectly plugPagNFCInfosResult =
mPlugPag.detectNfcCardDirectly(PlugPagNearFieldCardData.ONLY_M, 20);
            if (plugPagNFCInfosResult.getResult() != RET_OK) {
                emitter.onError(new PlugPagException("Cartão não identificado"));
                emitter.onComplete();
                return;
            }

            emitter.onNext(RET_WAITING_REMOVE_CARD);
            Thread.sleep(5000);
            int result = 0;
            if (plugPagNFCInfosResult.getCid() != null) {
                final PlugPagNFCDetectRemoveCard plugPagNFCDetectRemoveCard = new
PlugPagNFCDetectRemoveCard(PlugPagNearFieldRemoveCardData.REMOVE,
plugPagNFCInfosResult.getCid());

                result =
mPlugPag.detectNfcRemoveDirectly(plugPagNFCDetectRemoveCard);
                if (result != RET_OK) {
                    emitter.onError(new PlugPagException("Cartão não removido"));
                    emitter.onComplete();
                    return;
                }
            }

            emitter.onNext(result);
            //Starta a antena NFC
            mPlugPag.stopNFCCardDirectly();
        } catch (Exception e) {
            e.printStackTrace();
            emitter.onError(e);
        }

        emitter.onComplete();
    });
}
```


detectJustAuthDirectly

```
public Observable<Integer> detectJustAuthDirectly() {
    // chave de autenticacao de um cartao NFC virgem
    byte[] key = { (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte) 0xFF, (byte)
0xFF, (byte) 0xFF };
    return Observable.create(emitter -> {
        try {
            // Starta a antena NFC
            int resultStartNfc = mPlugPag.startNFCCardDirectly();
            PlugPagNFCInfosResultDirectly plugPagNFCInfosResult =
mPlugPag.detectNfcCardDirectly(PlugPagNearFieldCardData.ONLY_M, 20);

            if (plugPagNFCInfosResult.getResult() != RET_OK) {
                emitter.onError(new PlugPagException("Cartão não identificado"));
                emitter.onComplete();
                return;
            }

            final PlugPagNFCAuthDirectly auth = new PlugPagNFCAuthDirectly((byte)
0, key, EM1KeyType.TYPE_B, plugPagNFCInfosResult.getSerialNumber());
            int resultAuth = mPlugPag.justAuthNfcDirectly(auth);

            if (resultAuth != RET_OK) {
                emitter.onError(new PlugPagException(String.format("Erro ao
autenticar bloco")));
                emitter.onComplete();
                return;
            }

            emitter.onNext(resultAuth);
            // Stopa a antena NFC
            mPlugPag.stopNFCCardDirectly();
        } catch (Exception e) {
            e.printStackTrace();
            emitter.onError(e);
        }

        emitter.onComplete();
    });
}
```

Comandos APDU

OBS: A leitura do cartão NFC é sempre necessária. Após detectar o cartão, poderá enviar os comandos APDU. Sempre deixar o cartão próximo ao leitor.

```
public Observable<PlugPagCmdExchangeResult> cmdExchange() {
    return Observable.create(emitter -> {
        // Comando APDU
        byte[] command = new byte[]{
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00,
            (byte) 0x00
        };
        try {
            PlugPagCmdExchangeResult resultAuth =
mPlugPag.apduCommand(command, 256);

            if (resultAuth != null && resultAuth.getCmd() != null &&
resultAuth.getCmd().length > 0) {
                emitter.onNext(resultAuth);
            }else{
;                emitter.onError(new PlugPagException("Ocorreu um erro no
comando APDU."));
            }

            emitter.onComplete();
        } catch (Exception e) {
            e.printStackTrace();
            emitter.onError(e);
        }
    });
}
```

Pagamento de R\$250,00, no QR CODE Crédito à vista:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_QRCODE_CREDITO,  
            25000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Pagamento de R\$300, no QR Code Crédito, parcelado vendedor em 3 parcelas:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_QRCODE_CREDITO,  
            30000,  
            PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR,  
            3,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);  
  
        // Trata o resultado da transação  
        ...  
    }  
}
```

Pagamento de R\$300, no QR Code Crédito, parcelado comprador em 3 parcelas:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =
```

```

        new PlugPagPaymentData(
            PlugPag.TYPE_QRCODE_CREDITO,
            30000,
            PlugPag.INSTALLMENT_TYPE_PARC_COMPRADOR,
            3,
            "CODVENDA");

    // Cria a identificação do aplicativo
    PlugPagAppIdentification appIdentification =
        new PlugPagAppIdentification("MeuApp", "1.0.7");

    // Cria a referência do PlugPag
    PlugPag plugpag = new PlugPag(context, appIdentification);

    // Ativa terminal e faz o pagamento
    int initResult = plugpag.initializeAndActivatePinpad(new
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));

    if (initResult == PlugPag.RET_OK) {
        PlugPagTransactionResult result = plugpag.doPayment(paymentData);

        // Trata o resultado da transação
        ...
    }
}

```

Exemplos assíncronos

Pagamentos

Pagamento de R\$250,00, no crédito, à vista:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_CREDITO,  
            25000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);
```

```

// Ativa terminal e faz o pagamento
int initResult = plugpag.initializeAndActivatePinpad(new
    PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));

if (initResult == PlugPag.RET_OK) {
    mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener() {
        @Override
        public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
            //seu código aqui
        }

        @Override
        public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
            //seu código aqui
        }

        @Override
        public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
            //seu código aqui
        }

        @Override
        public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
            //seu código aqui
        }

        @Override
        public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
            //seu código aqui
        }
    });
}
}

```

Pagamento de R\$300, no crédito, parcelado em 3 parcelas:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_CREDITO,  
            30000,  
            PlugPag.INSTALLMENT_TYPE_PARC_VENDEDOR,  
            3,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener() {  
            @Override  
            public void onSuccess(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onError(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onPaymentProgress(@NotNull PlugPagEventData  
plugPagEventData) {  
                //seu código aqui  
            }  
        })  
    }  
}
```



```
    }

    @Override
    public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }

    @Override
    public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }
    });
}
}
```

Pagamento de R\$150,00, no débito:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_DEBITO,  
            15000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        mPlugPag.doAsyncPayment(paymentData, new PlugPagPaymentListener() {  
            @Override  
            public void onSuccess(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onError(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onPaymentProgress(@NotNull PlugPagEventData  
plugPagEventData) {
```

```
        //seu código aqui
    }

    @Override
    public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }

    @Override
    public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }
    });
}
}
```

Pagamento de R\$50, no voucher:

```
public void startPayment(Context context) {  
    // Define os dados do pagamento  
    PlugPagPaymentData paymentData =  
        new PlugPagPaymentData(  
            PlugPag.TYPE_VOUCHER,  
            5000,  
            PlugPag.INSTALLMENT_TYPE_A_VISTA,  
            1,  
            "CODVENDA");  
  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Ativa terminal e faz o pagamento  
    int initResult = plugpag.initializeAndActivatePinpad(new  
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));  
  
    if (initResult == PlugPag.RET_OK) {  
        plugpag.doAsyncPayment(paymentData, new PlugPagPaymentListener() {  
            @Override  
            public void onSuccess(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onError(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onPaymentProgress(@NotNull PlugPagEventData  
plugPagEventData) {  
                //seu código aqui  
            }  
        });  
    }  
}
```

```
    }

    @Override
    public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }

    @Override
    public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
        //seu código aqui
    }
    });
}
}
```

Estornar um pagamento

```
public void voidPayment(Context context) {
    // Cria a identificação do aplicativo
    PlugPagAppIdentification appIdentification =
        new PlugPagAppIdentification("MeuApp", "1.0.7");

    // Cria a referência do PlugPag
    PlugPag plugpag = new PlugPag(context, appIdentification);

    // Ativa terminal e faz o pagamento
    int initResult = plugpag.initializeAndActivatePinpad(new
        PlugPagActivationData("SEU_CODIGO_DE_ATIVAÇÃO"));

    if (initResult == PlugPag.RET_OK) {
        mPlugPag.doAsyncVoidPayment("transactionCode", "transactionId",
            new PlugPagPaymentListener() {
                @Override
                public void onSuccess(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
                    //seu código aqui
                }

                @Override
                public void onError(@NotNull PlugPagTransactionResult
plugPagTransactionResult) {
                    //seu código aqui
                }

                @Override
                public void onPaymentProgress(@NotNull PlugPagEventData
plugPagEventData) {
                    //seu código aqui
                }

                @Override
                public void onPrinterSuccess(@NotNull PlugPagPrintResult
plugPagPrintResult) {
                    //seu código aqui
                }
            }
        );
    }
}
```

```
        @Override
        public void onPrinterError(@NotNull PlugPagPrintResult
plugPagPrintResult) {
            //seu código aqui
        }
    });
}
}
```

Verificar autenticação

```
public void checkAuthentication(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Verifica autenticação  
    plugPag.asyncIsAuthenticated(new PlugPagIsActivatedListener() {  
        @Override  
        public void onIsActivated(boolean isActivated) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onError(String message) {  
            //seu código aqui  
        }  
    });  
}
```


Invaldar autenticação

```
public void logout(Context context) {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Invalida a autenticação existente  
    // Solicita autenticação  
    plugPag.asyncDeactivate(new PlugPagActivationData(activationCode),  
        new PlugPagActivationListener() {  
            @Override  
            public void onActivationProgress(@NotNull PlugPagEventData  
                plugPagEventData) {  
//seu código aqui  
            }  
  
            @Override  
            public void onSuccess(@NotNull PlugPagInitializationResult  
                plugPagInitializationResult) {  
                //seu código aqui  
            }  
  
            @Override  
            public void onError(@NotNull PlugPagInitializationResult  
                plugPagInitializationResult) {  
                //seu código aqui  
            }  
        });  
    ...  
}
```

Solicitar ativação

```
public void requestAuth(Activity activity) {
    // Cria a identificação do aplicativo
    PlugPagAppIdentification appIdentification =
        new PlugPagAppIdentification("MeuApp", "1.0.7");

    // Cria a referência do PlugPag
    PlugPag plugPag = new PlugPag(activity, appIdentification);

    // Cria objeto com informação do código de ativação
    PlugPagActivationData plugPagActivationData =
        new PlugPagActivationData("SeuCodigoDeAtivação");

    // Solicita autenticação
    plugPag.doAsyncInitializeAndActivatePinpad(
        new PlugPagActivationData(activationCode),
        new PlugPagActivationListener() {

            @Override
            public void onActivationProgress(@NotNull PlugPagEventData
                plugPagEventData) {

                //seu código aqui
            }

            @Override
            public void onSuccess(@NotNull PlugPagInitializationResult
                plugPagInitializationResult) {

                //seu código aqui
            }

            @Override
            public void onError(@NotNull PlugPagInitializationResult
                plugPagInitializationResult) {

                //seu código aqui
            }

        });
    ...
}
```

Reimpressão da via do estabelecimento

```
public void reprintStablishmentReceipt() {
    // Cria a identificação do aplicativo
    PlugPagAppIdentification appIdentification =
        new PlugPagAppIdentification("MeuApp", "1.0.7");

    // Cria a referência do PlugPag
    PlugPag plugpag = new PlugPag(context, appIdentification);

    // Reimprime a via do estabelecimento
    plugPag.asyncReprintEstablishmentReceipt(new PlugPagPrinterListener() {
        @Override
        public void onError(@NotNull PlugPagPrintResult plugPagPrintResult) {
            //seu código aqui
        }

        @Override
        public void onSuccess(@NotNull PlugPagPrintResult plugPagPrintResult) {
            //seu código aqui
        }
    });
}
```

Reimpressão da via do cliente

```
public void reprintCustomerReceipt() {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Reimprime via do cliente  
    plugPag.asyncReprintEstablishmentReceipt(new PlugPagPrinterListener() {  
        @Override  
        public void onError(@NotNull PlugPagPrintResult plugPagPrintResult) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onSuccess(@NotNull PlugPagPrintResult plugPagPrintResult) {  
            //seu código aqui  
        }  
    });  
}
```

Calcular parcelas

```
public void calculateInstallments() {  
    // Cria a identificação do aplicativo  
    PlugPagAppIdentification appIdentification =  
        new PlugPagAppIdentification("MeuApp", "1.0.7");  
  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context, appIdentification);  
  
    // Obtém a versão da biblioteca  
    plugpag.asyncCalculateInstallments("", new PlugPagInstallmentsListener() {  
        @Override  
        public void onCalculateInstallments(@NotNull String[] strings) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onError(@NotNull String s) {  
            //seu código aqui  
        }  
    });  
}
```

Ler cartão NFC

```
public void readNFCCard() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
    PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();  
    dataCard.setStartSlot(1);  
    dataCard.setEndSlot(1);  
  
    // Lê um cartão NFC  
    plugpag.asyncReadNFC(dataCard, new PlugPagNFCListener() {  
        @Override  
        public void onSuccess(@NotNull PlugPagNFCResult plugPagNFCResult) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onError(@NotNull String errorMessage) {  
            //seu código aqui  
        }  
    });  
}
```

Escrever no cartão NFC

```
public void writeToNFCCard() {
    // Cria a referência do PlugPag
    PlugPag plugpag = new PlugPag(context);

    String info = "teste_com16bytes";
    byte[] infoBytes = info.getBytes();

    PlugPagNearFieldCardData dataCard = new PlugPagNearFieldCardData();
    dataCard.setStartSlot(1);
    dataCard.setEndSlot(2);
    dataCard.getSlots()[1].put("data", infoBytes);

    // Escreve em um cartão NFC
    plugpag.asyncWriteNFC(dataCard, new PlugPagNFCListener() {
        @Override
        public void onSuccess(@NotNull PlugPagNFCResult plugPagNFCResult) {
            //seu código aqui
        }

        @Override
        public void onError(@NotNull String errorMessage) {
            //seu código aqui
        }
    });
}
```

Abortar operação de leitura/escrita no cartão NFC

```
public void abortNFC() {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    // Escreve em um cartão NFC  
    plugpag.asyncAbortNFC(new PlugPagAbortListener() {  
        @Override  
        public void onAbortRequested(boolean b) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onError(@NotNull String s) {  
            //seu código aqui  
        }  
    });  
}
```


Buscar última transação aprovada

```
public void getLastApprovedTransaction(Context context) {  
    // Cria a referência do PlugPag  
    PlugPag plugpag = new PlugPag(context);  
  
    plugpag.doAsyncVoidPayment(new PlugPagLastTransactionListener() {  
        @Override  
        public void onRequestedLastTransaction(@NotNull PlugPagTransactionResult  
plugPagTransactionResult) {  
            //seu código aqui  
        }  
  
        @Override  
        public void onError(@NotNull String s) {  
            //seu código aqui  
        }  
    });  
}
```

Utilizar métodos mock que vão alterar no PlugPag através do wrapper

```
/* Nesse exemplo estamos utilizando o Coroutines para realizar chamadas
assíncronas, mas pode ser utilizado Rx também, como você já vem utilizando em
outros exemplos. Ter uma dependência do PlugPag em uma classe */
class MockManager(private val plugPag: PlugPag) {
    // Método assíncrono
    suspend fun setMockState(hasToEnable: Boolean) = try {
        plugPag.setMockState(hasToEnable)
    } catch (e: Exception) { null }

    fun setMockType(typeSelected: Int) = try {
        plugPag.setMockResult(typeSelected)
    } catch (e: Exception) { null }

    suspend fun getMockState() = try {
        plugPag.getMockState()
    } catch (e: Exception) { null }

    suspend fun getMockResult() = try {
        plugPag.getMockResult()
    } catch (e: Exception) { null }
}
```

Utilizar métodos mock para realizar um pagamento

Esses método que realiza a chamada do **pagamento**.

```
suspend fun actionPayment() = withContext(Dispatchers.IO) {
    //ativa o mock
    mock.setMockState(MockManager.ENABLE MOCK)

    //seleciona tipo de retorno
    mock.setMockType(SUCCESS_CODE)
```

```

        //faz o pagamento
        operations.doPayment()

        //desativa o mock
        mock.setMockState(MockManager.DISABLE MOCK)
    }

    // Método que realiza a ativação.
    suspend fun doActivation() = withContext(Dispatchers.IO) {
        //ativa o mock
        mock.setMockState(MockManager.ENABLE MOCK)

        //seleciona tipo de retorno
        mock.setMockType(SUCCESS_CODE)

        //faz a ativação
        operations.doActivation()

        //desativa o mock
        mock.setMockState(MockManager.DISABLE MOCK)
    }

    // Método que realiza o estorno.
    suspend fun doRefund() = withContext(Dispatchers.IO) {
        countPassword = 0
        //ativa o mock
        mock.setMockState(MockManager.ENABLE MOCK)

        //seleciona tipo de retorno
        mock.setMockType(SUCCESS_CODE)

        //faz o estorno
        operations.doRefund()

        //desativa o mock
        mock.setMockState(MockManager.DISABLE MOCK)
    }
}

```

Códigos de retorno

Os códigos de retorno descritos abaixo são obtidos ao chamar o método `getResult()` de um `PlugPagTransactionResult` retornado por um dos métodos de transação de um objeto `PlugPag`: `doPayment(PlugPagPaymentData)`, `voidPayment(PlugPagVoidData)` e `getLastApprovedTransaction()`.

Valor	Descrição	Ação
0	Transação concluída com sucesso.	
-1001	Mensagem gerada maior que buffer dimensionado.	Coletar log (se existir) e enviar para o suporte.
-1002	Parâmetro de aplicação inválido.	Coletar log (se existir) e enviar para o suporte.
-1003	Terminal não está pronto para transacionar.	Tente novamente.
-1004	Transação não realizada.	Verificar mensagem retornada.
-1005	Buffer de resposta da transação inválido ao obter as informações de resultado da transação.	Realizar consulta de última transação.
-1006	Parâmetro de valor da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1007	Parâmetro de valor total da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1008	Parâmetro de código de venda não pode ser nulo.	Verificar implementação da chamada da biblioteca.

-1009	Parâmetro de resultado da transação não pode ser nulo.	Verificar implementação da chamada da biblioteca.
-1010	Driver de conexão não encontrado.	Verificar se todos os arquivos estão no diretório correto.
-1011	Erro ao utilizar driver de conexão.	Reinstalar os arquivos do driver de conexão.
-1012	Formato do valor da venda inválido.	Valor deve ser um número inteiro sem vírgula.
-1013	Comprimento do código de venda superior a 10 dígitos.	Truncar código de venda para no máximo 10 dígitos.
-1014	Buffer de recepção corrompido.	Refaça a transação.
-1015	Nome da aplicação maior que 25 caracteres.	Limitar nome da aplicação a 25 caracteres.
-1016	Versão da aplicação maior que 10 caracteres.	Limitar versão da aplicação em 10 caracteres.
-1017	Necessário definir nome da aplicação.	Definir nome e versão da aplicação com <code>setVersionName(String, String)</code>
-1018	Não existem dados da última transação.	Refaça a transação.
-1019	Erro de comunicação com terminal (resposta inesperada).	Realizar consulta de última transação.
-1024	Erro na carga de tabelas.	Refazer inicialização (carga de tabelas).

-1030	Token não encontrado	Refazer autenticação.
-1031	Valor inválido	Verificar o valor configurado para pagamento e tentar novamente. Valor mínimo: R\$ 1,00
-1032	Parcelamento inválido	Verificar o número de parcelas e tentar novamente.
-2001	Porta COM informada não encontrada.	Informar uma porta COM válida.
-2002	Não foi possível obter configurações da porta COM informada.	Informar uma porta COM válida.
-2003	Não foi possível configurar a porta COM informada.	Informar uma porta COM válida.
-2005	Não foi possível enviar dados pela porta COM informada.	Informar uma porta COM válida.
-2022	Java – Adaptador Null.	Verificar implementação.
-2023	Java – erro em DeviceToUse.	Coletar log (se existir) e enviar para o suporte.
-2024	Java – erro no serviço RfcommSocket.	Coletar log (se existir) e enviar para o suporte.
-2026	Java – Close exception.	Coletar log (se existir) e enviar para o suporte.
-3001	Permissão de root	Remover permissão de root do aparelho.
-4046	Não existe dados de autenticação	Efetuar a autenticação.

Códigos de erro de impressão

Os códigos de retorno descritos abaixo são obtidos ao chamar o método `getResult()` de um `PlugPagTransactionResult` retornado por um dos métodos de transação de um objeto `PlugPag`: `doPayment(PlugPagPaymentData)`, `voidPayment(PlugPagVoidData)` e `getLastApprovedTransaction()`.