DT 2

INTEGRAÇÃO COM A SMARTPOS STONE

Guia prático

TIAGO ALEXANDRE DOS SANTOS
DT COMPONENTES

INTRODUÇÃO

Bem-vindo ao nosso pequeno manual! Este guia é um resumo prático das informações essenciais disponíveis no portal oficial, que você pode acessar em sdkandroid.stone.com.br. Nosso objetivo é apresentar de forma clara e concisa os principais pontos, facilitando o seu entendimento e aplicação dos conceitos relacionados.

Este documento contém uma explicação detalhada da unidade untSTONEIntent, com foco na classe TStoneTEF. Aqui, abordaremos cada aspecto dessa classe, linha por linha, explicando os conceitos envolvidos, incluindo a criação e destruição de objetos, além do uso de intents. A ideia é oferecer uma visão prática e objetiva para otimizar seu uso dos recursos e funcionalidades.

Para uma visão mais aprofundada, recomendamos que consulte o portal diretamente. Esperamos que este material seja útil para você!

ESTRUTURA GERAL DO ARQUIVO

O arquivo começa com a declaração da unidade e a inclusão de bibliotecas que fornecem funcionalidades de interface gráfica, controle de eventos e integração com dispositivos Android:

```
unit untSTONEIntent;

interface

uses
System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs, FMX.Objects,
FMX.Controls.Presentation, FMX.StdCtrls, FMX.Platform
{$IFDEF ANDROID}
,Androidapi.JNI.GraphicsContentViewText,
Androidapi.Helpers,
System.Messaging,
Androidapi.JNI.App,
Androidapi.JNI.Net,
FMX.Platform.Android,
Androidapi.JNI.Os,
Androidapi.JNI.JavaTypes
{$ENDIF}
;
```

`unit`: Define o início de uma unidade em Delphi, o que é equivalente a um módulo ou arquivo de código em outros contextos.

`uses`: Importa diversas bibliotecas e módulos. Por exemplo, `FMX.Types` e `FMX.Forms` são usados para componentes gráficos e formulários no FireMonkey. Já `Androidapi` importa APIs Java usadas quando compilamos para Android.

`{$IFDEF ANDROID}`: Uma diretiva condicional que só inclui os módulos dentro dela se o código for compilado para a plataforma Android.

CONSTANSTES DA CLASSE

As constantes definidas dentro da classe são usadas para padronizar valores e facilitar a leitura do código:

```
const
 RETURN_SCHEME                = 'stone-delphi-demo';
 RETURN_SCHEME_CANCEL         = 'stone-delphi-demo-cancel';

 TRANSACAO_CREDITO             ='CREDIT';
 TRANSACAO_DEBITO              = 'DEBIT';
 TRANSACAO_PIX                 = 'PIX';
 TRANSACAO_VOUCHER             = 'VOUCHER';
 TRANSACAO_INSTANT_PAYMENT     = 'INSTANT_PAYMENT';

 CREDITO_A_VISTA               = 'NONE';    // A VISTA
 CREDITO_PARC_SEM_JUROS        = 'MERCHANT'; // PARCELADO SEM JUROS
```

- **`RETURN_SCHEME` e `RETURN_SCHEME_CANCEL`:** URLs usadas para identificar o retorno de ações na aplicação após a conclusão ou cancelamento de uma transação.
- **Tipos de Transação (`TRANSACAO_CREDITO`, `TRANSACAO_DEBITO`, etc.):** Estes constantes definem os tipos de transações que podem ser realizadas, como crédito, débito, PIX, voucher, etc.
- **Tipos de Crédito (`CREDITO_A_VISTA`, `CREDITO_PARC_SEM_JUROS`):** Definem como os pagamentos a crédito são realizados, se à vista ou parcelado.

DEFINIÇÃO DA CLASSE TSTONETEF

A classe `TStoneTEF` é responsável pela integração com um sistema de pagamentos (TEF - Transferência Eletrônica de Fundos). Abaixo está a definição inicial da classe:

```
type TStoneTEF = class
```

A classe `TStoneTEF` é declarada aqui. Em Delphi, uma classe é uma estrutura de dados que pode conter campos, métodos e eventos. Vamos agora analisar os campos privados e métodos dessa classe.

ATRIBUTOS PRIVADOS

Os campos privados são variáveis internas da classe, que armazenam informações usadas em todo o processo de transação de pagamento:

```
 private
   FOnResultadoRecebido: TNotifyEvent;
   FEerro: String;
   FPan: string;
   FCode: string;
   FAmount: string;
   FBrand: string;
   FxMessage: string;
   FAuthorization_code: string;
   FAuthorizationcode: string;
   Fatk: string;
```

- **`FOnResultadoRecebido`:** Este é um evento que será disparado quando um resultado for recebido da transação. Eventos em Delphi seguem o padrão de delegação de eventos do tipo `TNotifyEvent`.
- **`FErro`:** Armazena a mensagem de erro, caso ocorra durante o processo de pagamento.
- **`FPan`,`FCode`,`FAmount`,`FBrand`:** Esses campos armazenam informações como o PAN (Primary Account Number), código da transação, valor e marca do cartão.
- **`FxMessage`:** Uma mensagem extra de transação que pode ser usada para logs ou depuração.
- **`FAuthorization_code`e `FAuthorizationcode`:** Códigos de autorização da transação, emitidos pelo sistema de pagamento.
- **`Fatk`:** Possivelmente um token de autenticação.

METODOS PUBLICOS

Os métodos públicos da classe estão descritos abaixo. Eles são utilizados para iniciar transações e tratar eventos relacionados ao ciclo de vida de atividades Android.

```
public
 {$IFDEF ANDROID}
   procedure startPayment(transactionType, amount, editable_amount,
installment_type, installment_count, order_id: string);
   procedure startCancel(transactionType, amount, editable_amount,
installment_type, installment_count, order_id: string);

   procedure HandleActivityMessage(const Sender: TObject; const M: TMessage);
   function HandleAppEvent(AAppEvent: TApplicationEvent; AContext: TObject):
Boolean;
   function HandleIntentAction(const Data: JIntent): Boolean;

   constructor Create(AOwner: TComponent);
   destructor Destroy;
 {$ENDIF}
```

- **`startPayment`:** Inicia um processo de pagamento, aceitando parâmetros como o tipo de transação, valor, número de parcelas e o ID do pedido.
- **`startCancel`:** Inicia o processo de cancelamento de uma transação, utilizando parâmetros semelhantes aos do método `startPayment`.
- **`HandleActivityMessage`:** Manipula mensagens que são passadas entre diferentes atividades (telas) no Android. Usado para receber respostas ou notificações de eventos da aplicação.
- **`HandleAppEvent`:** Monitora eventos do ciclo de vida do aplicativo, como quando o aplicativo entra em segundo plano ou é fechado.
- **`HandleIntentAction`:** Este método lida diretamente com `Intents`. Um **Intent** em Android é uma estrutura que permite a comunicação entre diferentes componentes do sistema, como atividades ou serviços. Aqui, ele processa intents relacionados à ação do pagamento.
- **`constructor Create`:** Método construtor da classe, responsável por inicializar os objetos e seus atributos.
- **`destructor Destroy`:** Método destrutor, que libera a memória alocada e outros

PROCEDURES E FUNCTIONS DA CLASSE TSTONETEF

Este documento contém explicações detalhadas apenas das procedures e functions da classe `TStoneTEF`, especificadas no bloco `public`. Cada uma delas será comentada em detalhes abaixo.

```
public
 {$IFDEF ANDROID}
   procedure startPayment(transactionType, amount, editable_amount,
installment_type, installment_count, order_id: string);
   procedure startCancel(transactionType, amount, editable_amount,
installment_type, installment_count, order_id: string);

   procedure HandleActivityMessage(const Sender: TObject; const M: TMessage);
   function HandleAppEvent(AAppEvent: TApplicationEvent; AContext: TObject):
Boolean;
   function HandleIntentAction(const Data: JIntent): Boolean;

   constructor Create(AOwner: TComponent);
   destructor Destroy;
 {$ENDIF}
```

PROCEDURE STARTPAYMENT

Esta procedure é responsável por iniciar um processo de pagamento. Os parâmetros recebidos incluem o tipo de transação (crédito, débito, etc.), o valor da transação, se o valor pode ser editado, o tipo de parcelamento, o número de parcelas e o identificador do pedido (order_id). Estes parâmetros configuram a transação antes de ser enviada para processamento.

```
procedure startPayment(transactionType, amount, editable_amount,
installment_type, installment_count, order_id: string);
```

PROCEDURE STARTCANCEL

Esta procedure é semelhante à `startPayment`, mas ao invés de iniciar uma nova transação, ela cancela uma transação existente. Os parâmetros são os mesmos: tipo de transação, valor, tipo de parcelamento, etc., e servem para identificar a transação a ser cancelada.

```
procedure startCancel(transactionType, amount, editable_amount, installment_type,
installment_count, order_id: string);
```

PROCEDURE HANDLEACTIVITYMESSAGE

Este método trata mensagens de atividades que ocorrem no Android, como notificações entre diferentes atividades (telas). O método é chamado quando um evento do sistema ou da aplicação gera uma mensagem que precisa ser tratada. O parâmetro `Sender` indica a origem da mensagem, e `M` é a mensagem em si. Isso é útil para lidar com eventos externos que influenciam no fluxo da aplicação, como o retorno de um resultado de pagamento.

```
procedure HandleActivityMessage(const Sender: TObject; const M: TMessage);
```

FUNCTION HANDLEAPPEVENT

Esta função monitora eventos no ciclo de vida do aplicativo, como quando o aplicativo é suspenso (minimizado) ou retomado. O parâmetro `AAppEvent` descreve o tipo de evento que ocorreu (por exemplo, `TApplicationEvent.BecameActive` quando o app volta a ser ativo), e `AContext` fornece um contexto adicional. Esta função retorna um valor booleano indicando se o evento foi tratado com sucesso.

```
function HandleAppEvent(AAppEvent: TApplicationEvent; AContext: TObject): Boolean;
```

FUNCTION HANDLEINTENTACTION

Essa função lida com **Intents** no Android. Um Intent é uma estrutura que permite a comunicação entre diferentes componentes, como atividades ou serviços. O parâmetro `Data` representa o Intent que contém informações da ação que foi executada. A função retorna um valor booleano indicando se o Intent foi tratado corretamente. Isso é comumente utilizado para lidar com ações externas ou retornos de outras atividades, como a conclusão de uma transação.

```
function HandleIntentAction(const Data: JIntent): Boolean;
```

CONSTRUCTOR CREATE

O construtor `Create` é o ponto onde a classe é inicializada. Aqui, os atributos da classe são configurados e qualquer inicialização necessária é feita. O parâmetro `AOwner` especifica quem é o dono ou controlador desse objeto (geralmente um formulário ou outro componente).

```
constructor Create(AOwner: TComponent);
```

DESTRUCTOR DESTROY

O destrutor `Destroy` é responsável por liberar qualquer recurso ou memória alocada pela classe quando ela não for mais necessária. Este método é chamado automaticamente quando o objeto é destruído, garantindo que não haverá vazamento de memória.

```
destructor Destroy;
```

Encerramos por aqui nosso pequeno manual, esperando que ele tenha servido como um guia claro e prático para entender e aplicar os conceitos abordados. Nosso objetivo foi fornecer uma explicação detalhada e acessível da unidade untSTONEIntent e da classe TStoneTEF, auxiliando você a utilizar esses recursos de maneira eficiente.

Lembre-se de que o portal oficial **sdkandroid.stone.com.br** está sempre à disposição para consultas mais aprofundadas e atualizações. Agradecemos por acompanhar este material, e desejamos sucesso na sua jornada com a API!