Datatrans iOS Payment Library

Developer's Manual

Attention! Important changes in iOS 10.

- Do not forget to add NSCameraUsageDescription to your info.plist when credit card scanning is enabled.

Attention!

- Important changes in iOS 9. Please read section 4.1.
- If you are unable to archive your app, please refer to section 7.1.

Datatrans AG
Swiss E-Payment Competence
Kreuzbühlstrasse 26, 8008 Zürich, Switzerland
Tel. +41 44 256 81 91, Fax +41 44 256 81 98
www.datatrans.ch

Revisions

Version	Date	Author	Comment
0.1	2010-09-17	Basil Achermann ieffects ag	First draft
0.2	2010-09-20	Arman Mkrtchyan	Changed logos, replaced Figure 2.1 and added Figure 4.1
0.2	2010-09-21	Arman Mkrtchyan	Added logo to first page, changed old datatrans logo to the new one, changed Figure 2.2 (skiped last 2 screens), deleted property "localizedMerchant" from class DTPaymentRequest.
0.3	2010-09-21	Basil Achermann	DTVisualStyle API changes
1.0	2010-09-27	Basil Achermann	Styling chapter updated, DTVisualStyle API changes, 3-D secure added to screenshots
1.1	2010-10-21	Arman Mkrtchyan	Added "Secure payment by Datatrans" to screenshots
1.2	2010-10-25	Basil Achermann	Library 1.1 changes, API clarifications
1.3	2011-02-16	Basil Achermann	Library 1.2 changes
1.4	2011-03-11	Basil Achermann	Library 1.3 changes
1.4.1	2011-04-11	Basil Achermann	Dutch localization added
1.5	2011-05-12	Basil Achermann	Rotation and landscape mode
1.6	2012-09-21	Basil Achermann	iOS 6, iPad support, card holder optional, accessibility
1.6.1	2012-09-25	Basil Achermann	iOS 6 issues
1.6.2	2012-09-25	Basil Achermann	Auto-fill issue
1.7	2013-06-06	Basil Achermann	Recurring payments (PostFinance Card & PayPal)
1.7.1	2013-08-19	Basil Achermann	DTPaymentReturnsCreditCardAlways
1.8	2013-10-23	Basil Achermann	Hidden mode with card data, MyOne, iOS 7
1.8.1	2014-02-13	Basil Achermann	New payment options, 64-bit support
1.8.2	2014-07-03	Basil Achermann	Certificate pinning option
1.8.3	2014-08-13	Basil Achermann	Swisscom Easypay added
1.9.0	2014-09-16	Basil Achermann	New alias generation modes, iOS 8
1.9.1	2014-11-25	Basil Achermann	PostFinance Card registration
2.0.0	2015-03-15	Basil Achermann	Easypay alias + ELV
2.1.0	2015-04-02	Basil Achermann	Card scanning
2.1.1	2015-04-16	Basil Achermann	Web-based CC input option
2.1.2	2015-05-04	Basil Achermann	Language option
2.2.0	2015-07-17	Basil Achermann	SwissBilling payment method added
2.3.0	2015-10-16	Basil Achermann	JCB added, iOS 9 changes
2.4.0	2015-10-29	Basil Achermann	TWINT added
2.5.0	2016-06-28	Basil Achermann	Keyboard fix, suppress error option, Bitcode error
2.6.0	2016-07-06	Basil Achermann	Apple Pay added
2.7.0	2016-07-14	Basil Achermann	TWINT registration/alias support

2.7.1	2016-08-19	Patrick Schmid	TWINT adjustments (refno)
2.7.2	2016-09-09	Basil Achermann	Bugfixes

Table of Contents

1 Introduction

Datatrans AG, leading Swiss payment service provider, has developed Datatrans iOS Payment Library (DTiPL). DTiPL allows application developers to use Datatrans AG's credit card payment services natively on iPhones and iPads. This manual provides guidance on library installation, invocation, and other issues of importance to developers who wish to integrate DTiPL into their mobile applications.

1.1 Document Structure

Chapter 1 – Introduction
Explains this document's structure and content.

Chapter 2 – Overview
Gives an overview of the Datatrans iOS Payment Library.

Chapter 3 – Key Concepts
Explains key concepts of DTiPL and discusses some of the most common use cases.

Chapter 4 – API
Contains detailed API documentation.

Chapter 5 – Integration
Explains library installation and integration into Xcode.

1.2 Scope

This document provides information on using DTiPL to create mobile commerce apps on iPhone and iPad devices. As such, it is primarily aimed at developers of iOS applications.

It is assumed that the reader is already familiar with Datatrans AG's products and services. Also, knowledge of the Objective-C programming language, UIKit, as well as basic understanding of Xcode are required. Covering these topics is beyond the scope of this document.

1.3 Conventions

Throughout this document, the following styles are used:

Name
Emphasized technical terms, organization/product names

Path
File system paths, file names etc.

Class
Class and method names

```
void codeSample() {
   code(); // sample code
}
```

Code listings

<replaceable>
Text meant to be replaced with data by the developer

2 Overview

2.1 Payment Methods

The library currently supports the following cards: VISA, MasterCard, Diners Club, American Express, JCB, and Manor MyOne. Additionally, PayPal, PostFinance Card/ E-Finance, Swisscom Easypay, German Lastschriftverfahren (ELV), SwissBilling, and TWINT are supported.

2.2 Supported Platforms

Apple devices with iOS 6.0 or higher are supported. The library has been localized for English, French, German, Italian, and Dutch.

2.3 Library Tasks

The payment library is responsible for the following tasks:

- Validation: credit card number, expiration date and CVV are validated online.
- Authentication: if merchant and credit card are enrolled with 3-D Secure services, authentication ensures that the card is being used by its legitimate owner.
- Authorization: if amount, currency are valid and within the card's limit, the payment transaction is authorized and can be completed by the merchant once goods are being delivered (settlement process).

2.4 Payment Process

Figure 2-1 gives an overview of the shopping and payment process on the iOS device.

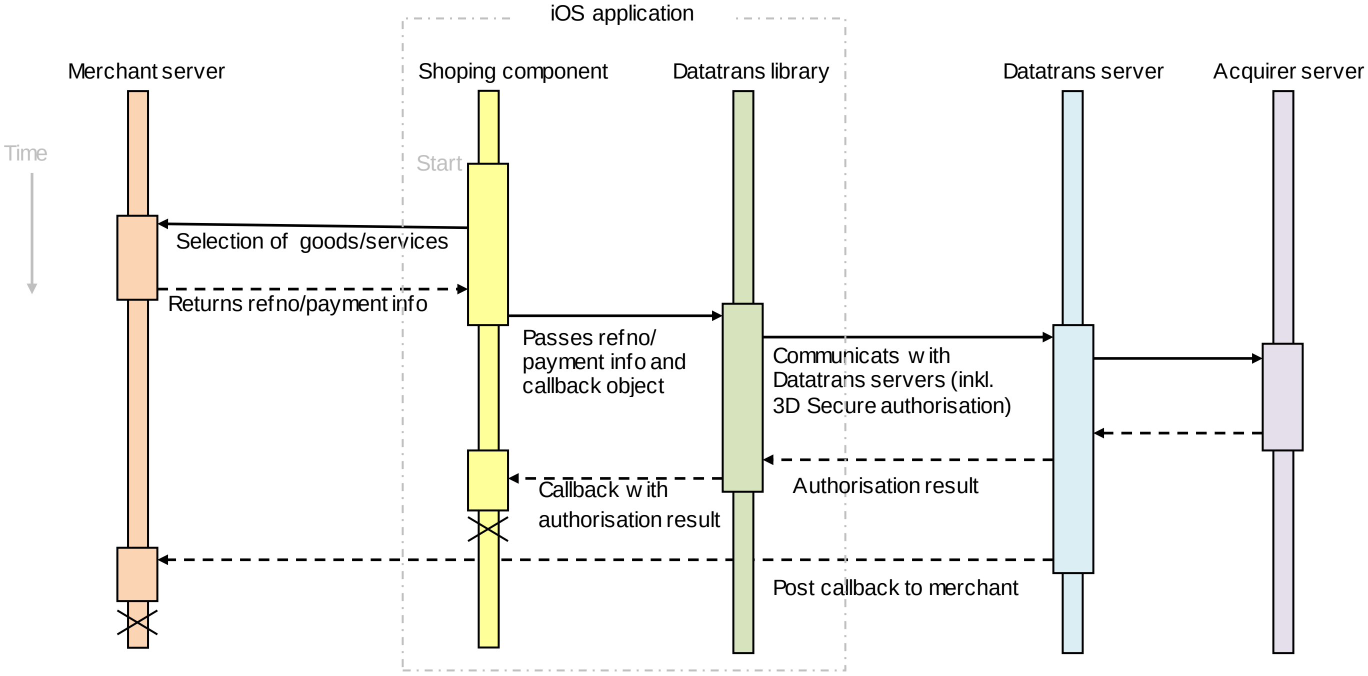

iOS application
Merchant server
Shoping component
Datatrans library
Datatrans server
Acquirer server
Time
Start
Selection of goods/services
Returns refno/payment info
Passes refno/ payment info and callback object
Communicats with Datatrans servers (inkl. 3D Secure authorisation)
Callback with authorisation result
Authorisation result
Post callback to merchant

Figure 2-1: Payment process overview

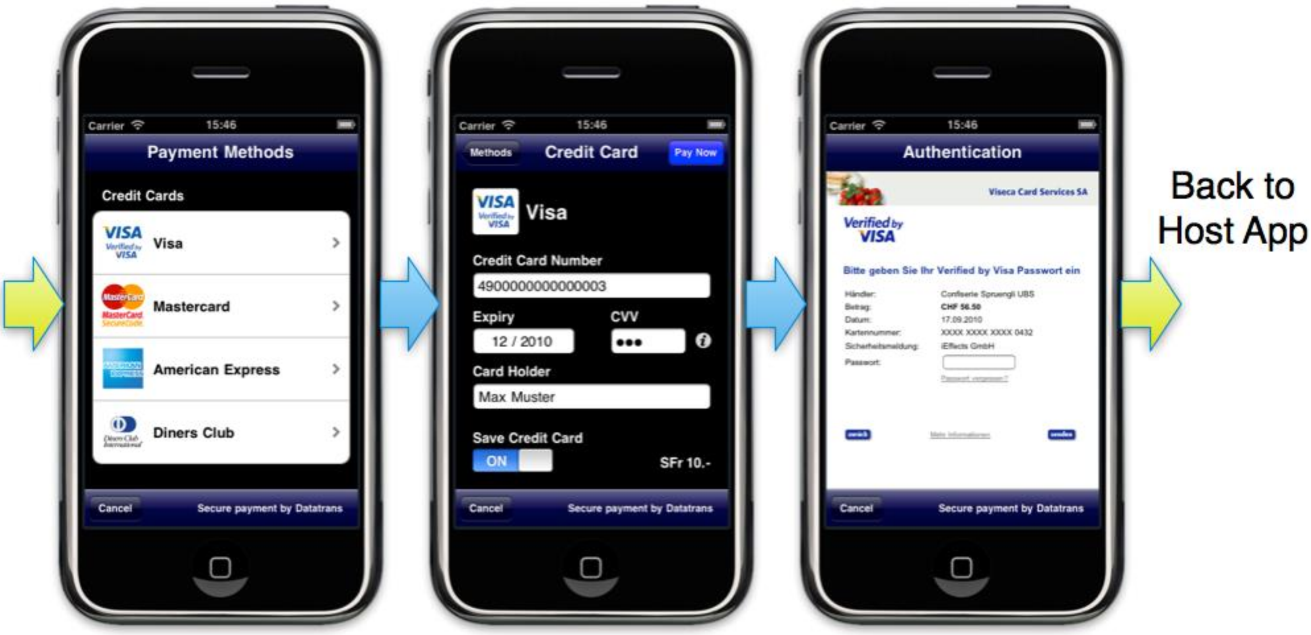
The following steps occur during a successful session:

1. Host app: user selects goods/services to buy from a merchant. When the user proceeds to checkout, complete order information is sent to the merchant's server. In return, the app receives a transaction reference number (refno).

2. App passes payment information and refno to DTiPL.
3. In a series of network calls and user interactions, the library performs all necessary steps to authenticate the user (including 3-D Secure) and authorize the purchase.
4. Transaction is authorized in the background.
5. When authorization is completed, the merchant's server is informed by Datatrans AG’s server. The previously supplied refno (see step 1) is used to identify and execute the order.
6. App control is given back to the main app component via callback.

2.5 User Interface

Figure 2-2 shows how the payment process is presented to the app user. The library can be invoked with or without payment method selection. If a credit card has been used for a previous order, an alias can be supplied to directly proceed to authentication and/or authorization steps (first two screens skipped). An app may also choose to implement its own payment method selection. In this case, the first screen is not displayed.



Credit Card Number: 4900000000000003
Expiry: 12 / 2010
CVV: ●●●
Card Holder: Max Muster
Save Credit Card: ON
SFr 10.-

Bitte geben Sie Ihr Verified by Visa Passwort ein

Händler: Confiserie Spruengli UBS
Betrag: CHF 56.50
Datum: 17.09.2010
Kartennummer: XXXX XXXX XXXX 0432
Sicherheitsmeldung: iEffects GmbH
Passwort:

Figure 2-2: Library screen shots

3 Key Concepts

3.1 DTPaymentController

The library's core component is the DTPaymentController class. It must be invoked on an existing UINavigationConroller instance. The payment controller pushes its own view controllers on top of existing ones. The navigation bar is used to navigate back and forth between library screens.

At the bottom of the screen, a toolbar with a cancel button is displayed. Previous toolbar/navigation bar items are kept and restored when the payment controller is dismissed.

When the payment process is finished, a delegate method is invoked to put the app back in control. At this point, the app may choose to push additional view controllers (e. g. thank you screen) or just dismiss the payment view controller.

This navigation-based design as well as customizable colors and fonts allow payment to be put into a bigger checkout process unobtrusively, and make it look like an integral part of the host application.

3.2 Library Invocation

Prior to library invocation, the host app must obtain a unique transaction reference number (refno) to identify the order. This is typically done by sending complete order information (basket contents, shipping information etc.) to the merchant's web server. The server generates a refno that is stored along with the order and sends it back to the mobile device. Optionally, the server also returns the HMAC-MD5 signature for additional payment security.

The library is invoked with refno, merchant ID, and pricing information. Alternatively, the library can be invoked in hidden mode. In hidden mode, credit card information is also supplied by the app. The payment method selection screens are then skipped and authentication and/or authorization take place immediately.

Listing 3-1 shows an example of how DTiPL is invoked in standard mode.

```
DTPaymentRequest* paymentRequest = [[DTPaymentRequest alloc] init];
paymentRequest.amountInSmallestCurrencyUnit = 1000;
paymentRequest.currencyCode = @"CHF";
paymentRequest.localizedPriceDescription = @"CHF 10.-";
paymentRequest.merchantId = @"12345";
paymentRequest.refno = @"refno12345";

NSArray* paymentMethods = [DTPaymentController allAvailablePaymentMethods];

DTPaymentController* paymentController = [DTPaymentController
                                           paymentControllerWithDelegate:self
                                           paymentRequest:paymentRequest
                                           paymentMethods:paymentMethods];

// set some options here... (see next chapter)

[paymentController presentInNavigationController:self.navigationController
                   animated:YES];
```

Listing 3-1: DTPaymentController invocation in standard mode

Some notes:

- Payment methods can be adjusted to include only methods supported by the merchant, e.g. for merchants without Diner’s Club contract, DTPaymentMethodDinersClub should not be present in the paymentMethods array.
- Default styles are used in this example, see section 3.6 for customized styles.
- No signature is used in this example.

3.3 Options

The payment controller can be configured with a number of options. For example, if the library is supposed to connect to the Datatrans test system instead of production servers, the `testing` option must be enabled. Listing 3.2 shows a sample configuration.

In this example:

- the library is invoked in test mode
- a back button is displayed in the upper left corner of the start screen
- the library shows the credit card holder field, but doesn’t force the user to fill it in
- the library tries to acquire an alias for future PostFinance Card payments

See section 5.10 for a list of all options.

```
...
DTPaymentController* paymentController = [DTPaymentController
                                            paymentControllerWithDelegate:self
                                            paymentRequest:paymentRequest
                                            paymentMethods:paymentMethods];

paymentController.paymentOptions.testing = YES;
paymentController.paymentOptions.showBackButtonOnFirstScreen = YES;
paymentController.paymentOptions.cardHolder = DTPaymentCardHolderOptional;
paymentController.paymentOptions.returnsPostFinanceAlias = YES;

[paymentController presentInNavigationController:self.navigationController
                     animated:YES];
...
```

Listing 3-2: DTPaymentOptions example

3.4 App Callback Notification

The app must register a DTPaymentControllerDelegate delegate with the payment controller. The delegate is notified when payment is finished (success, error, or user cancellation).

After successful payment, the app can retrieve payment method information from the payment controller. This information can be stored for future payments in hidden mode.

Credit card aliases are returned depending on option `returnsCreditCard`. For PayPal, PostFinance Card, ELV and Easypay aliases, options `returnsPayPalAlias` , `returnsPostFinanceAlias`, `returnsELVAlias`, `returnsEasypayAlias` must be enabled respectively.

Listing 3-3 contains a delegate notification code sample. Please note:

- It is impossible to securely store credit card information on the device or server without prompting the user for his or her password every time. However, considerable effort is necessary to access data on an iPhone/iPad and even more so to understand how this data was stored on the device. It is therefore acceptable to store the alias in encrypted form on the iOS device if the user gives permission to do so.

- In the example, the payment controller is simply dismissed, meaning that the screen before DTiPL invocation reappears. Typically, the app would push yet another screen (success screen) on top of the last view controller and then remove everything using [controller.naviationController popToRootViewControllerAnimated:YES] or pop to some other view, i.e. last view before the checkout (not payment) process.

```
- (void)paymentControllerDidFinish:(DTPaymentController *)controller {
   if (controller.recurringPaymentMethod != nil) {
      DTRecurringPaymentMethod* recurring = controller.recurringPaymentMethod;
      // store recurring payment details securely on server...
      NSString* alias = recurring.alias;
      if ([recurring isKindOfClass:DTCreditCard.class]) {
         DTCreditCard* cc = (DTCreditCard *)recurring;
         NSString* holder = cc.cardHolder;
         NSString* maskedCC = cc.maskedCC;
         //...
      } else if ([recurring isKindOfClass:DTPostFinanceCard.class]) {
         DTPostFinanceCard * pfc = (DTPostFinanceCard *)recurring;
         NSString* maskedCC = pfc.maskedCC;
         //...
      } else if ([recurring isKindOfClass:DTPayPal.class]) {
         DTPayPal* pp = (DTPayPal *)recurring;
         NSString* email = pp.email;
         //...
      }

      // or serialize the payment method and store locally...
      NSData* data = [recurring data];
      // store data encrypted on device (app responsible for encryption)
      // Use [DTRecurringPaymentMethod recurringPaymentMethodWithData:data]
      // to deserialize.
   }
   [controller dismissAnimated:YES];
}
```

Listing 3-3: Delegate notification on success

3.5 Merchant Notification

On successful authorization, Datatrans AG's authorization server invokes the merchant's postURL as defined by field URL Post in Datatrans Web Admin. Among other information, fields shown in Listing 3-4 are posted as form post or XML post. The merchant's web server retrieves payment information previously stored with the same refno and matches currency code and amount. It then executes the order and performs transaction settlement with Datatrans using the returned authorizationCode value.

```
amount=1000
currency=CHF
refno=refno12345
uppTransactionId=1009161410129l5292
acqAuthorizationCode=982889
authorizationCode=915285337
```

Listing 3-4: postURL fields

3.6 UI Customization

Many colors and fonts used by payment views are customizable. For this purpose, a DTVisualStyle object can be set on the payment controller as shown in Listing 3-5. In this example, only the background color is set. For a conclusive list of display options, see API section 5.11.

```
DTPaymentController* paymentController = [DTPaymentController
                                           paymentControllerWithDelegate:self
                                           paymentRequest:paymentRequest
                                           paymentMethods:paymentMethods];

DTVisualStyle* style = [DTVisualStyle defaultStyle];
style.backgroundColor = [UIColor blackColor];

paymentController.visualStyle = style;
```

Listing 3-5: Applying custom style

Note that DTVisualStyle does not cover navigation bar color and toolbar color. These colors must be set on the app's UINavigationController directly or controlled with UIAppearance.

3.7 Hidden mode payments

In hidden mode, no payment selection takes place in the library. The app has to provide a recurring payment method (alias) from a previous transaction or payment method registration (see section 3.8), or complete card data as entered into the app's own payment selection screen. Note that for security reasons, **card number and CVV must not be stored by the app under any circumstances**! If no payment can take place at the moment of data entry, credit card data has to be discarded, or, at the very least, an alias has to be created. Listing 3-6 shows an invocation of the payment controller with a recurring payment method.

```
    // aliasPaymentMethod from previous transaction
    DTRecurringPaymentMethod* aliasPaymentMethod = ...;

    DTPaymentController* pc;
    pc = [DTPaymentController paymentControllerWithDelegate:self
                                             paymentRequest:paymentRequest
                                     recurringPaymentMethod:aliasPaymentMethod];
    [pc presentInNavigationController: navigationController animated:YES];
```

Listing 3-6: Recurring payment in hidden mode

A MyOne sample payment in hidden mode with raw card data is shown in Listing 3-7.

```
DTCardPaymentMethod* card = [[DTCardPaymentMethod alloc]
                              initWithPaymentMethod:DTPaymentMethodMyOne
                                             number:@"6004520200668702072"
                                           expMonth:12
                                            expYear:2015
                                                cvv:@"123"
                                             holder: @"Max Muster"];
DTPaymentController* pc;
pc = [DTPaymentController paymentControllerWithDelegate:self
                                         paymentRequest:paymentRequest
                                      cardPaymentMethod:card];
```

```
  [pc presentInNavigationController:self.navigationController animated:YES];
```

Listing 3-7: Hidden mode payment with card data

3.8 Payment method registration (alias request)

The library supports creating credit card, PostFinance Card, Easypay, TWINT and ELV alias numbers without making a payment. Aliases are allowed to be stored by the app and can be used for future hidden mode payments.

When creating an alias, the app can either use its own card input screen and pass the data to the library or let the library manage payment method selection.

3.8.1 Payment method selection/input by library (standard mode)

In this mode, the library's input screens are used to gather data for alias generation. DTPaymentOptions and DTVisualStyle options can be used to control test/production mode and cell styling. Credit card data is automatically verified in this mode with a test authorization of a small amount.

Listing 3-8 shows creation of a credit card alias in testing mode. The app is notified when the alias is available, see Listing 3-9.

```
  NSArray* paymentMethods = [NSArray arrayWithObjects:DTPaymentMethodVisa,
                             DTPaymentMethodMyOne, DTPostFinanceCard, nil];

  DTAliasRequest* ar = [[DTAliasRequest alloc] initWithMerchantId:merchantId
                                               paymentMethods:paymentMethods];

  DTPaymentController* pc = [DTPaymentController
                            paymentControllerWithDelegate:self aliasRequest:ar];

   pc.paymentOptions.testing = YES;
   pc.paymentOptions.showBackButtonOnFirstScreen = YES;
  [pc presentInNavigationController:self.navigationController animated:YES];
```

Listing 3-8: Creation of credit card alias in standard mode

```
- (void)paymentControllerDidFinish:(DTPaymentController *)controller {
      // the same as with regular payments, alias payment method stored in
      // controller.recurringPaymentMethod property
}
```

Listing 3-9: Alias notification

3.8.2 Card input by app (hidden mode)

In this mode, the library is invoked with the necessary credit card data. The library generates an alias and optionally verifies the given credit card with a test authorization transaction.

Listing 3-10 shows creation of a credit card alias in testing mode with verification. The app is notified as usual via the delegate. Note that this example will fail because the given credit card data is not valid. The same request would succeed with `verifyingTransaction:NO`.

```
 DTCardPaymentMethod* card = [[DTCardPaymentMethod alloc]
      initWithPaymentMethod:DTPaymentMethodVisa number:@"4444333322221111"
                   expMonth:12 expYear:2015 cvv:@"123" holder:nil];

 DTAliasRequest* ar = [[DTAliasRequest alloc] initWithMerchantId:merchantId
                cardPaymentMethod:card verifyingTransaction:YES];

DTPaymentController* pc = [DTPaymentController
                           paymentControllerWithDelegate:self aliasRequest:ar];

 pc.paymentOptions.testing = YES;
 [pc presentInNavigationController:self.navigationController animated:YES];
```

Listing 3-10: Creation of credit card alias in hidden mode

3.9 Error Handling

There are three kinds of errors:

- Technical errors: network interruption, memory or I/O errors
- Business errors: 3-D authentication failure, authorization failure
- Mistakes: typo or missing field

3.9.1 Technical Errors

The library is built with the policy that recoverable technical errors lead to non-fatal error messages. The user is lead to the previous screen and encouraged to try again.

3.9.2 Business Errors

The policy for business errors is that the payment process is aborted immediately.

3.9.3 Mistakes

Mistakes are caught by the app if easily possible (plausibility checks). The user is given the possibility to make a correction. Everything else is treated as business error.

3.10 Accessibility

The library supports Apple's *Accessibility* feature for people with disabilities and for automated UI testing. Controls have their accessibility label set to their title text. Credit card input fields are labeled as seen in Table 3-1.

Accessibility label	Description
`Credit Card Number`	Credit card number text field (`UITextField`)
`Expiry Date`	Credit card expiration date text field
`CVV Code`	CVV2/CVC2 code text field
`Card Holder`	Credit card holder text field
`Save Credit Card`	Save credit card number switch (`UISwitch`)

Table 3-1: Accessibility labels

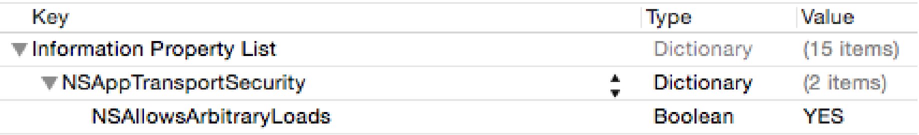
4 Mandatory settings

For some payment methods a number of configuration steps are required, otherwise payment transactions will fail.

4.1 Credit Cards / Swisscom Easypay (iOS 9)

Several third-party servers for credit card payments (3-D Secure) and Swisscom Easypay do not comply yet with iOS 9's new App Transport Security (ATS) requirements. **We recommend to temporarily allow such connections in the app's info.plist**. Please see Figure 4-1 for details.

Note that this setting does not make your app any less secure than it was under iOS 8.



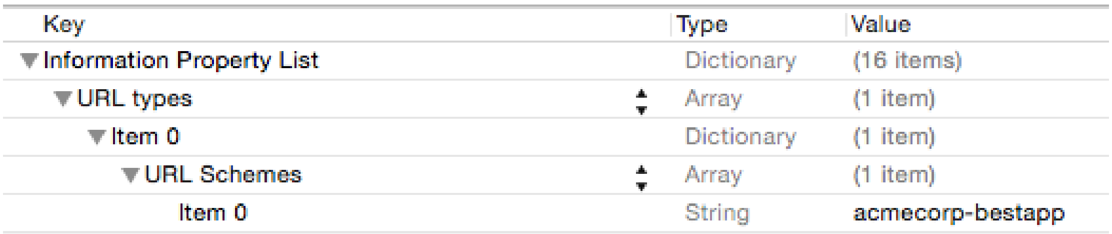
Key	Type	Value
▼Information Property List	Dictionary	(15 items)
▼NSAppTransportSecurity	Dictionary	(2 items)
NSAllowsArbitraryLoads	Boolean	YES

Figure 4-1: Mandatory app info.plist settings on iOS 9

4.2 TWINT

The TWINT app on the user's device has to call back into the shopping app during TWINT transactions. In order to do this, a URL scheme has to be defined in the app's info.plist (Figure 4-2) ***and*** configured via `DTPaymentOptions.twintAppCallbackScheme` (Listing 4-1).

Please note that there is no need to define a new scheme just for TWINT. Just set the `twintAppCallbackScheme` option if you already have a scheme defined. However, keep in mind that the scheme must be unique to the shopping app. Do not use actual protocols or file types such as "http", "mailto", "pdf" etc., generic names like "ticket", and especially do not use "twint".



Key	Type	Value
▼Information Property List	Dictionary	(16 items)
▼URL types	Array	(1 item)
▼Item 0	Dictionary	(1 item)
▼URL Schemes	Array	(1 item)
Item 0	String	acmecorp-bestapp

Figure 4-2: Application URL scheme definition

```
paymentController.paymentOptions.twintAppCallbackScheme = @"acmecorp-bestapp";
```

Listing 4-1: TWINT URL scheme option

4.3 SwissBilling

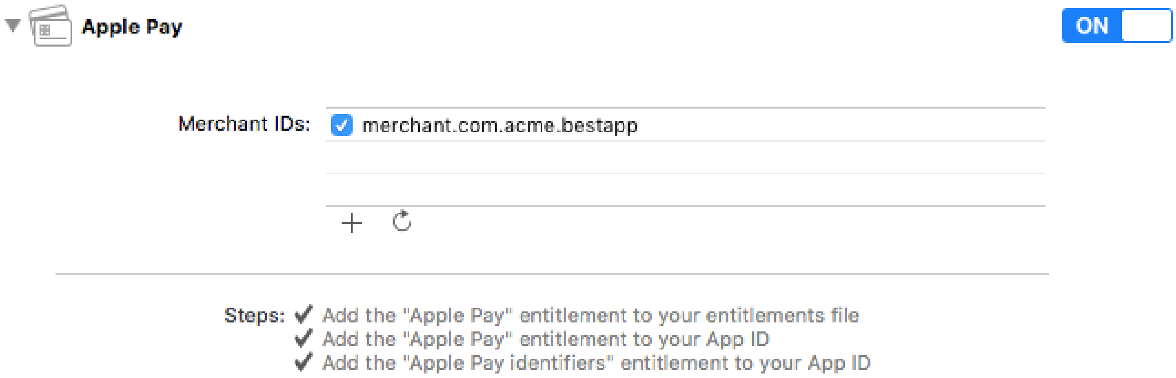
For SwissBilling transactions, a `DTSwissBillingPaymentInfo` (see section 5.15) has to be created and configured via `DTPaymentOptions.swissBillingPaymentInfo` (Listing 4-2).

```
paymentController.paymentOptions.swissBillingPaymentInfo = mySwissBillingInfo;
```

Listing 4-2: SwissBilling payment

4.4 Apple Pay

In order to use Apple Pay in your App you need to register a merchant ID in your developer account's Certificates, Identifiers & Profiles section. Your app then needs to be configured with Apple Pay capabilities (`Target -> Capabilities -> Apple Pay`) and entitlements. Figure 4-3 shows what the configuration should look like in Xcode.



Apple Pay ON

Merchant IDs: ☑ merchant.com.acme.bestapp

Steps: ✓ Add the "Apple Pay" entitlement to your entitlements file
✓ Add the "Apple Pay" entitlement to your App ID
✓ Add the "Apple Pay identifiers" entitlement to your App ID

Figure 4-3: Apple Pay Capability

The payment library then has to be configured with this merchant ID. There are several possibilities to use the library as described in the following sections.

4.4.1 Regular Payment

If you want to use Apple Pay just like any other payment method, you only have to provide your registered merchant ID as shown in Listing 4-3 and put `DTPaymentMethodApplePay` into the list of payment methods that you want to support.

Note that the library determines whether Apple Pay is present and hides that payment method if the device is not configured for Apple Pay.

```
DTPaymentController* controller = [DTPaymentController ...
DTApplePayConfig* ap = [[DTApplePayConfig alloc]
                        initWithMerchantIdentifier:@"merchant.com.acme.bestapp"];
controller.paymentOptions.applePayConfig = ap;
```

Listing 4-3: Configure Apple Pay for payments

4.4.2 Configuration Options

Apple Pay can be configured to request additional information from users, such as the shipping address or shipping type. It can also be set up to show more information than just the payment total. These settings are configured via `PKPaymentRequest` object (see official Apple Pay documentation for more information). The request object can be obtained and configured as shown in Listing 4-4. In this example, users are required to enter their postal and email addresses.

```
DTApplePayConfig* ap = [[DTApplePayConfig alloc]
                        initWithMerchantIdentifier:@"merchant.com.acme.bestapp"];
ap.request.requiredShippingAddressFields = PKAddressFieldPostalAddress |
                                           PKAddressFieldEmail;
controller.paymentOptions.applePayConfig = ap;
```

Listing 4-4: More Apple Pay configuration options

4.4.3 Interactive Payment

If you want to have full control over the Apple Pay process, you can register a `DTApplePayDelegate` object. This allows you to respond interactively to a user's actions. For example, you can calculate a new payment total based on the selected payment method or you can add an additional fee for the selected shipping method.

In the example in Listing 4-5 two shipping methods are configured and the delegate set.

```
DTApplePayConfig* ap = [[DTApplePayConfig alloc]
                         initWithMerchantIdentifier:@"merchant.com.acme.bestapp"];

PKShippingMethod* sm1 = [[PKShippingMethod alloc] init];
sm1.identifier = @"N";
sm1.label = @"Free shipping";
sm1.detail = @"Ships within 48 hours";
sm1.amount = [NSDecimalNumber decimalNumberWithString:@"0.00"];

PKShippingMethod* sm2 = [[PKShippingMethod alloc] init];
sm2.identifier = @"E";
sm2.label = @"Express delivery";
sm2.detail = @"Delivered within 24 hours";
sm2.amount = [NSDecimalNumber decimalNumberWithString:@"10.00"];

ap.request.shippingMethods = [NSArray arrayWithObjects:sm1, sm2, nil];
ap.delegate = self;

controller.paymentOptions.applePayConfig = ap;
```

Listing 4-5: Set the DTApplePayDelegate for interactive updates

The delegate's `didSelectShippingMethod` method is invoked when a user chooses or changes the shipping method. The new payment total (new summary items) can then be calculated based on the selected method (Listing 4-6).

```
- (void)paymentAuthorizationViewController:(PKPaymentAuthorizationViewController
      *)controller didSelectShippingMethod:(PKShippingMethod *)shippingMethod
                          completion:(void (^)(PKPaymentAuthorizationStatus,
         NSArray<PKPaymentSummaryItem *> *))completion {
     NSMutableArray* summaryItems = [NSMutableArray array];
     ... // create new summary items based on 'shippingMethod' and call completion
     completion(PKPaymentAuthorizationStatusSuccess, summaryItems);
}
```

Listing 4-6: Update summary items / payment total

`DTApplePayDelegate`'s delegate methods are taken directly from Apple's `PKPaymentAuthorizationViewControllerDelegate` definition and behave in the exact same way. For more information, please consult the official Apple Pay documentation.

4.4.4 Apple Pay Button

If you want to use a stand-alone Apple Pay button in your app, please do so by following Apple's guidelines. Once the user has pressed the button, configure the payment library as described above and set Apple Pay as the sole accepted payment method (Listing 4-7). Apple Pay will then start directly without additional library screens.

```
NSArray* paymentMethods = [NSArray arrayWithObject:DTPaymentMethodApplePay];
DTPaymentController* c = [DTPaymentController paymentControllerWithDelegate:self
                              paymentRequest:request paymentMethods:paymentMethods];
// additional Apple Pay configurations as explained above...
```

Listing 4-7: Direct invocation of Apple Pay

5 API

This chapter contains the library class reference. Each Objective-C class is presented in its own section.

Figure 4-5-1 gives an overview of the library's classes.

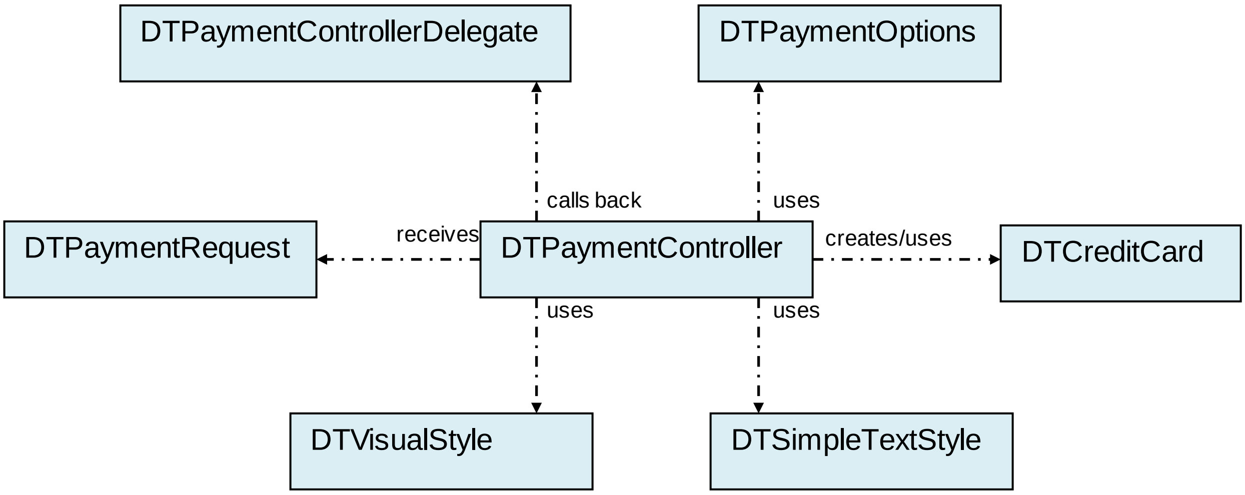


Figure 4-5-1: Library classes

5.1 DTPaymentController

The DTPaymentController class is the library's main entry point, see sections 3.1 and 3.2. The payment controller displays credit card selection, authentication, and authorization screens and does all the necessary network calls in the background. The payment controller must be invoked on an existing UINavigationConroller instance.

5.1.1 Class Methods

```
+ (id)paymentControllerWithDelegate:(id<DTPaymentControllerDelegate>)delegate
                paymentRequest:(DTPaymentRequest *)request
                paymentMethods:(NSArray *)methods
```

Creates and returns a new DTPaymentController object. The controller will display a payment method selection screen with the given payment methods.

Parameters

delegate

The delegate to receive success/error/cancellation notification.

request

The object containing payment information such as merchant ID, currency, amount, refno etc.

methods

An array of payment method string constants, i.e. one or more of:
DTPaymentMethodVisa, DTPaymentMethodMasterCard,
DTPaymentMethodDinersClub, DTPaymentMethodAmericanExpress, etc.

If only one method is supplied, the payment selection screen is skipped. This allows for external payment method selection.

Payment methods can be visually grouped. To achieve this, add lists of payment methods to the methods array. Only one level of grouping is supported, see Listing 5-1 for example.

```
NSArray* creditCards = [NSArray arrayWithObjects:DTPaymentMethodVisa,
                          DTPaymentMethodMasterCard,
                          DTPaymentMethodAmericanExpress,
                          DTPaymentMethodDinersClub,
                          nil];
NSArray* paypal = [NSArray arrayWithObjects:DTPaymentMethodPayPal, nil];
NSArray* paymentMethods = [NSArray arrayWithObjects:creditCards, paypal, nil];
```

Listing 5-1: Grouping payment methods

Returns

a newly created and initialized DTPaymentController object in standard mode.

```
+ (id)paymentControllerWithDelegate:(id<DTPaymentControllerDelegate>)delegate
                paymentRequest:(DTPaymentRequest *)request
                cardPaymentMethod:(DTCardPaymentMethod *)cardPaymentMethod;
```

Creates and returns a new DTPaymentController object in hidden mode with card information entered by the user. The controller will not display payment method selection/entry screens and proceed to authentication/authorization directly.

Parameters

delegate
: The delegate to receive success/error/cancellation notification.

request
: The object containing payment information such as merchant ID, currency, amount, refno etc.

cardPaymentMethod
: Raw credit card information eneterd by the user in a previous step.

Returns

a newly created and initialized DTPaymentController object in hidden mode.

```
+ (id)paymentControllerWithDelegate:(id<DTPaymentControllerDelegate>)delegate
                paymentRequest:(DTPaymentRequest *)request
                recurringPaymentMethod:(DTRecurringPaymentMethod *)recurringPM;
```

Creates and returns a new DTPaymentController object in hidden mode. The controller will not display payment method selection/entry screens and proceed to authentication/authorization directly.

Parameters

delegate
: The delegate to receive success/error/cancellation notification.

request
: The object containing payment information such as merchant ID, currency, amount, refno etc.

recurringPaymentMethod

Recurring payment information previously obtained from the controller after a successful transaction, see sections 3.4 and 5.5.

Returns

a newly created and initialized DTPaymentController object in hidden mode.

```
+ (id)paymentControllerWithDelegate:(id<DTPaymentControllerDelegate>)delegate
                 aliasRequest:(DTAliasRequest *)aliasRequest
```

Creates and returns a new DTPaymentController object for alias generation only (no payment).

Parameters

delegate

The delegate to receive success/error/cancellation notification.

aliasRequest

Alias request for standard/hidden mode alias generation.

Returns

a newly created and initialized DTPaymentController object.

```
+ (NSArray *)allAvailablePaymentMethods
```

Returns all available payment method constants.

Returns

an array of available payment method string constants.

5.1.2 Instance Methods

```
- (void)presentInNavigationController:animated
```

Shows the payment controller.

Parameters

controller

The navigation controller used to push payment view controllers.

animated

YES if view controllers are to be pushed animated, NO otherwise.

```
- (void)dismissAnimated:(BOOL)animated
```

Removes payment view controllers from the payment controller’s navigation controller.

Parameters

animated

YES if view controllers are to be popped animated, NO otherwise.

Attention: Do not perform view controller actions while an animation is taking place. For example dismissAnimated:YES immediately followed by pushViewController:animated: will cause rendering issues. Only animate the last action.

5.1.3 Properties

```
@property (nonatomic, retain) DTVisualStyle* visualStyle
     Display options (colors and fonts) for payment views. Optional property.

@property (nonatomic, copy) DTPaymentOptions* paymentOptions
     Options unrelated to display style. Optional property.

@property (nonatomic, readonly) DTRecurringPaymentMethod* recurringPaymentMethod
     Credit card or PF/PayPal alias information for future use in hidden mode. The
     property is only available (not nil) after successful payment or alias
     request.

@property (nonatomic, readonly) NSString* transactionId
     The ID of the last transaction if available. Call from
     paymentControllerDidFinish: or paymentController:didFailWithError:
```

5.2 DTPaymentControllerDelegate (protocol)

The DTPaymentControllerDelegate protocol is used to receive notifications from DTPaymentController. It must be implemented by the host application.

5.2.1 Class Methods

There are no class methods.

5.2.2 Instance Methods

```
- (void)paymentControllerDidFinish:(DTPaymentController *)controller
```

Invoked when authorization or alias generation has completed successfully. Alias payment method is available via the recurringPaymentMethod property.

Parameters

controller
    The payment controller responsible for this notification.

request
    The payment request that has been completed successfully.

```
- (void)paymentController:(DTPaymentController *)controller
             didFailWithError:(NSError *)error
```

Invoked when the payment transaction failed.

Parameters

controller
    The payment controller responsible for this notification.

error
    The error that has occurred. Possible error codes are specified in Table 5-1.

Error code	Description
DTPaymentErrorTechnical	Internal or technical error.
DTPaymentErrorValidation	Credit card information was invalid.
DTPaymentErrorAuthentication	Credit card holder could not be authenticated.
DTPaymentErrorAuthorization	Payment could not be authorized for the specified credit card.

Table 5-1: DTPaymentErrorCode codes

```
- (void)paymentController:(DTPaymentController *)controller
               didCancelWithType:(DTPaymentCancellationType)cancellationType;
```

Invoked when the payment transaction has been canceled by the user.

Parameters

controller

The payment controller responsible for this notification.

cancellationType

The reason why the transaction has been canceled. Possible values are specified in

Cancellation Type	Description
DTPaymentCancellationTypeBackButton	User pressed the back button on the first library screen. Only possible if back button is enabled via DTPaymentOptions.
DTPaymentCancellationTypeCancelButton	User pressed the Cancel button at the bottom of the screen.

Table 5-2: DTCancellationType types

```
- (BOOL) paymentController:(DTPaymentController*)controller
               shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
               orientation
```

Invoked when the device is rotated, see `shouldAutorotateToInterfaceOrientation` of class `UIViewController`. If this optional method is not implemented, only portrait mode is supported.

Parameters

orientation

The orientation of the application’s user interface after rotation. The possible values are described in `UIInterfaceOrientation`.

5.3 DTPaymentRequest

The DTPaymentRequest class describes a payment transaction. It contains information such as currency, amount, refno, merchant ID etc. The payment request is used when a new DTPaymentController is created.

5.3.1 Class Methods

There are no class methods.

5.3.2 Instance Methods

See properties.

5.3.3 Properties

```
@property (nonatomic, copy) NSString* merchantId
     The merchant ID, supplied by Datatrans AG.

@property (nonatomic, copy) NSString* refno
     The refno, supplied by the merchant’s server.

@property (nonatomic, copy) NSString* currencyCode
     The currency code (ISO 4217).

@property (nonatomic, assign) NSUInteger amountInSmallestCurrencyUnit
     The payment amount in the smallest unit of the given currency. For example,
     EUR 10 is represented as 1000, because one Euro is divided into 100 Euro
     cents, or, in other words, EUR is specified to have 2 digits after the decimal
     separator (ISO 4217).
```

```
@property (nonatomic, copy) NSString* signature
     The HMAC-MD5 signature of merchant ID, amount, currency, and refno using a
     shared secret between the merchant and Datatrans AG. Optional property.

@property (nonatomic, copy) NSString* localizedPriceDescription
     The price displayed on the credit card entry screen.
```

5.4 DTCardPaymentMethod

Class representing raw credit card data to be used for immediate payment. Apps use this class if they have their own user interface for payment method entry.

5.4.1 Class Methods

There are no class methods.

5.4.2 Instance Methods

```
-(id)initWithPaymentMethod:(NSString *)method number:(NSString *)number
        expMonth:(NSUInteger)expMonth  expYear:(NSUInteger)expYear
        cvv:(NSString *)cvv holder:(NSString *)holder;
```

Initalizes a DTCardPaymentMethod object with card data. See properties for description of values.

5.4.3 Properties

```
@property (nonatomic, copy) NSString* paymentMethod
     Payment method constant, e.g. DTPaymentMethodVisa.

@property (nonatomic, copy) NSString* number
     Card number.

@property (nonatomic, assign) NSUInteger expMonth
     Expiration month, [1, 12], e. g. 9 for September.

@property (nonatomic, assign) NSUInteger expYear
     Expiration year, 4 digits, e. g. 2010 for 2010.

@property (nonatomic, copy) NSString* cvv
     CVV string, nil for Diners Club cards.

@property (nonatomic, copy) NSString* holder
     Card holder’s name or nil.
```

5.5 DTRecurringPaymentMethod

Base class for recurring payment methods. This class contains only an alias string for future payments. Subclasses DTPostFinanceCard, DTPayPal, and DTCreditCard provide additional information e.g. the masked card number.

5.5.1 Class Methods

```
+ (id) recurringPaymentMethodWithData:(NSData *)data
```

Creates and returns a new recurring payment method object from a given NSData object. This is a convenience method for serialization/deserialization.

Note: This method deserializes to the correct subclass, i.e. a DTCreditCard object is returned if data contains a serialized credit card.

Parameters

data

Data to be deserialized into a recurring payment method object.

See also instance method -(NSData *) data.

5.5.2 Instance Methods

```
- (NSData *)data
```

Returns an NSData representation of the recurring payment method. This is a convenience method for serialization/deserialization. The data object is not encrypted.

See also class method +(id)recurringPaymentMethodWithData:(NSData *)data.

5.5.3 Properties

```
@property (nonatomic, copy) NSString* alias
     Alias string to be used for future payments.
```

5.6 DTCreditCard

The DTCreditCard class contains information about a credit card used in an earlier payment transaction. The DTCreditCard object is used to initialize a DTPaymentController in hidden mode, i.e. without payment method selection screen. It can be obtained by either making a successful credit card payment or by creating an alias.

5.6.1 Class Methods

```
+ (id)creditCardWithData:(NSData *)data
```

Creates and returns a new credit card from a given NSData object. This is a convenience method for serialization/deserialization.

Parameters

data

Data to be deserialized into a credit card object.

See also instance method -(NSData *) data.

5.6.2 Instance Methods

```
- (NSData *)data
```

Returns an NSData representation of the credit card. This is a convenience method for serialization/deserialization. The data object is not encrypted.

See also class method +(id)creditCardWithData:(NSData *)data.

5.6.3 Properties

```
@property (nonatomic, assign) NSUInteger expMonth
     Expiration month, [1, 12], e. g. 9 for September.

@property (nonatomic, assign) NSUInteger expYear
     Expiration year, 4 digits, e. g. 2010 for 2010.

@property (nonatomic, copy) NSString* maskedCC
     Masked credit card number for displaying purposes, e. g. 432930xxxxxx6095.

@property (nonatomic, copy) NSString* alias
     Credit card number alias. May be used interchangeably with the real credit
     card number for a given merchant.

@property (nonatomic, copy) NSString* cardHolder
     Card holder’s name.

@property (nonatomic, copy) NSString* paymentMethod
     Payment method constant, e.g. DTPaymentMethodVisa.
```

5.7 DTELV

Payment information for recurring ELV payments.

5.7.1 Class Methods

```
+ (id)elvWithData:(NSData *)data
```

Creates and returns a new DTELV object from a given NSData object. This is a convenience method for serialization/deserialization.

Parameters

data

Data to be deserialized into a DTELV object.

See also instance method -(NSData *)data.

5.7.2 Instance Methods

```
- (id)initWithAlias:(NSString *)alias
```

Init method with an alias string.

Parameters

alias

ELV alias string.

```
- (id)initWithAlias:(NSString *)alias bankrouting:(NSString *)routing
```

Init method to be used with old-style ELV aliases. Please use `initWithAlias:` with aliases returned by the library.

Parameters

alias

ELV alias string obtained before April 15 2015.

routing

Bank routing (Bankleitzahl) required for aliases created before April 15 2015.

```
- (NSData *)data
```

Returns an NSData representation of this method. This is a convenience method for serialization/deserialization. The data object is not encrypted.

See also class method +(id)elvWithData:(NSData *)data.

5.7.3 Properties

```
@property (nonatomic, copy) NSString* alias
     ELV alias for future payments.
```

5.8 DTPostFinanceCard

Payment information for recurring PostFinance Card payments.

5.8.1 Class Methods

```
+ (id)pfCardWithData:(NSData *)data
```

Creates and returns a new PostFinance card from a given NSData object. This is a convenience method for serialization/deserialization.

Parameters

data

Data to be deserialized into a DTPostFinanceCard object.

See also instance method -(NSData *)data.

5.8.2 Instance Methods

```
- (NSData *)data
```

Returns an NSData representation of the card. This is a convenience method for serialization/deserialization. The data object is not encrypted.

See also class method +(id)pfCardWithData:(NSData *)data.

5.8.3 Properties

```
@property (nonatomic, copy) NSString* maskedCC
     Masked post finance card number for displaying purposes, e.g. xxx xxx xxx 471.
```

```
@property (nonatomic, copy) NSString* alias
     PostFinance card alias for future payments.
```

5.9 DTPayPal

Payment information for recurring PayPal payments.

5.9.1 Class Methods

```
+ (id)ppWithData:(NSData *)data
```

Creates and returns a new PayPal object from a given NSData object. This is a convenience method for serialization/deserialization.

Parameters

data

Data to be deserialized into a DTPayPal object.

See also instance method -(NSData *)data.

5.9.2 Instance Methods

```
- (NSData *)data
```

Returns an NSData representation of the payment method. This is a convenience method for serialization/deserialization. The data object is not encrypted.

See also class method +(id)ppCardWithData:(NSData *)data.

5.9.3 Properties

```
@property (nonatomic, copy) NSString* email
     PayPal email address for displaying purposes.
```

```
@property (nonatomic, copy) NSString* alias
     PayPAl alias for future payments.
```

5.10 DTPaymentOptions

The DTPaymentOptions class is used for settings unrelated to visual appearance.

5.10.1 Class Methods

There are no class methods.

5.10.2 Instance Methods

See properties.

5.10.3 Properties

`@property (nonatomic, assign) BOOL testing`

Whether Datatrans AG's test or production server should be used. YES, if the test environment should be used, NO otherwise. Default value: NO.

`@property (nonatomic, assign) DTPaymentReturnsCreditCard returnsCreditCard`

Whether the app is interested in the user's credit card data for future recurring/alias transactions. Default value: DTPaymentReturnsCreditCardNever.

See Table 5-3 for a description of DTPaymentReturnsCreditCard constants.

Constant	Description
DTPaymentReturnsCreditCardNever	Credit card information is not returned and the user is not asked if credit card should be stored.
DTPaymentReturnsCreditCardSelectableDefaultNo	Credit card information is returned if the user gives permission to do so. The UISwitch is initially set to NO (don't store credit card).
DTPaymentReturnsCreditCardSelectableDefaultYes	Credit card information is returned if the user gives permission to do so. The UISwitch is initially set to YES (store credit card).
DTPaymentReturnsCreditCardAlways	Credit card information is always returned. The user is not asked whether data should be stored.

Table 5-3: DTPaymentReturnsCreditCard constants

`@property (nonatomic, assign) BOOL returnsPostFinanceAlias`

Whether PostFinance Card transactions should be invoked in recurring payment mode. If the user accepts the terms, a DTPostFinanceCard object will be returned for future payments upon successful completion.

`@property (nonatomic, assign) BOOL returnsPayPalAlias`

Whether PayPal transactions should be invoked in recurring payment mode. If the user accepts the terms, a DTPayPal object will be returned for future payments upon successful completion.

`@property (nonatomic, assign) BOOL returnsELVAlias`

Whether ELV transactions should be invoked in recurring payment mode. If the user accepts the terms, a DTELV object will be returned for future payments upon successful completion.

`@property (nonatomic, assign) BOOL returnsEasypayAlias`

Whether Easypay transactions should be invoked in recurring payment mode. If the user accepts the terms, a DTRecurringPaymentMethod object will be returned for future payments upon successful completion.

`@property (nonatomic, assign) BOOL displayShippingDetails`

Whether shipping details (address) should be visible when making a PayPal transaction in recurring payment mode (returnsPayPalAlias flag set). Default value: YES.

`@property (nonatomic, assign) BOOL showBackButtonOnFirstScreen`

Whether the first screen of the library should have its back button enabled. When the user presses this button, a cancel notification is sent to the app. Default value: NO.

`@property (nonatomic, assign) BOOL showAuthorizationConfirmationScreen`

Whether a confirmation screen should be shown before final authorization takes place. Default value: NO.

`@property (nonatomic, assign) NSDictionary* merchantProperties`

A set of merchant-defined key-value pairs of type NSString*. Properties are

```
     sent along with the payment request and posted to the merchant’s PostURL. Note
     that this option is only available for purely web-based payment methods, i.e.
     currently only PayPal and PostFinance methods.

@property (nonatomic, assign) BOOL useWebCreditCardInput
     Whether non-native credit card forms should be used. Default value: NO.

@property (nonatomic, assign) BOOL autoSettlement
     Whether the transaction should be settled automatically. Default value: NO.

@property (nonatomic, copy) NSString* easypayTitle
     Title used for Swisscom Easypay payments.

@property (nonatomic, copy) NSString* easypayDescription
     Description used for Swisscom Easypay payments.

@property (nonatomic, copy) NSString* easypayPaymentInfo
     Payment info used for Swisscom Easypay payments.

@property (nonatomic, assign) BOOL easypayPresentedAsNATELPay
     Whether Swisscom Easypay should be displayed as Swisscom NATEL Pay instead.
     Default value: NO.

@property (nonatomic, copy) NSString* creditCardInputLocalizedDoneButtonTitle
     Override title for the done button on the credit card input screen (default:
     “Pay” or “Proceed”).

@property (nonatomic, assign) BOOL certificatePinning
     Whether secure connections to Datatrans servers require a certificate chain
     signed with a specific CA private key. The device’s trust settings are
     explicitly ignored, i.e. custom installed/white-listed certificates and/or CAs
     will not work.

     Please be advised that enabling this option will break your app in many
     corporate networks with anti-malware/-theft/-espionage SSL proxying.

     Default value: NO.

@property (nonatomic, copy) NSString* language
     Library language override, ISO 639-1 two-letter code, e.g. “de” or “en”.

     Default value: nil (use device language)

@property (nonatomic, copy) DTSwissBillingPaymentInfo* swissBillingPaymentInfo
     Payment information for a SwissBilling transaction.

@property (nonatomic, copy) NSString* twintAppCallbackScheme
     Unique URL scheme used by the TWINT app to call the merchant app.

@property (nonatomic, assign) BOOL suppressBusinessErrorDialog
     Do not present an error dialog if a critical/business error occurs. Default
     value: NO.

@property (nonatomic, assign) BOOL skipAuthorizationCompletion
     Skip the last step of the authorization process for external/manual
     authorization. Default:NO

@property (nonatomic, copy) DTApplePayConfig* applePayConfig
     Configuration object, mandatory for Apple Pay transactions.

@property (nonatomic, assign) BOOL creditCardScanningEnabled
     Whether a scan button should be shown on the credit card entry screen. Default
     value: YES.

@property (nonatomic, assign) DTPaymentCardHolder cardHolder

     Whether the cardholder’s name is required. See Table 5-4 for a description of
     DTPaymentCardHolder constants.
```

Constant	Description
`DTPaymentCardHolderHidden`	The credit cardholder field is hidden. This is the default setting.
`DTPaymentCardHolderOptional`	The credit card holder field is visible but the value is optional.

`DTPaymentCardHolderRequired`	The credit card holder’s name is required.

Table 5-4: DTPaymentCardHolder constants

5.11 DTVisualStyle

The DTVisualStyle class is used for all settings related to visual appearance. Classes DTSimpleTextStyle and DTShadowTextStyle are used for most style settings (see sections 5.12 and 5.13).

5.11.1 Class Methods

```
+ (DTVisualStyle *)defaultStyle;
```

Creates and returns the default display style. Changes can then be applied selectively using properties.

5.11.2 Instance Methods

See properties.

5.11.3 Properties

All properties are optional if the object is created using +(DTVisualStyle *)defaultStyle.

```
@property (nonatomic, retain) UIColor* backgroundColor
     Screen background color.

@property (nonatomic, copy) DTShadowTextStyle* navigationBarTitleStyle
     Font and color of navigation bar text.

@property (nonatomic, retain) UIColor* navigationBarButtonItemDoneTintColor
     Color of navigation bar done/pay button.

@property (nonatomic, copy) DTShadowTextStyle* titleStyle
     Font and color of text titles.

@property (nonatomic, copy) DTShadowTextStyle* textStyle
     Font and color of regular text.

@property (nonatomic, copy) DTShadowTextStyle* emphasizedTextStyle
     Font and color of emphasized text.

@property (nonatomic, copy) DTShadowTextStyle* tableViewTitleStyle
     Font and color of table view group titles.

@property (nonatomic, copy) DTShadowTextStyle* tableViewCellTextStyle
     Font and color of table view cells.

@property (nonatomic, copy) DTSimpleTextStyle* inputFieldStyle
     Font and color of input fields.

@property (nonatomic, copy) DTShadowTextStyle* inputLabelStyle
     Font and color of input field labels.

@property (nonatomic, assign) BOOL isDark
     YES if the color scheme is dark, NO otherwise. An info button of type
     UIButtonTypeInfoLight is drawn if color scheme is dark, UIButtonTypeInfoDark
     otherwise. (< iOS 7 only).
```

5.12 DTSimpleTextStyle

The DTSimpleTextStyle class defines a text’s font and color.

5.12.1 Class Methods

There are no class methods.

5.12.2 Instance Methods

See properties.

5.12.3 Properties

```
@property (nonatomic, retain) UIColor* foregroundColor
     Text color.

@property (nonatomic, retain) UIFont* font
     Text font.
```

5.13 DTShadowTextStyle

The DTShadowTextStyle class inherits from DTSimpleTextStyle and adds drop shadow definitions.

5.13.1 Class Methods

There are no class methods.

5.13.2 Instance Methods

See properties.

5.13.3 Properties

```
@property (nonatomic, retain) UIColor* shadowColor
     Shadow color. Optional property.

@property (nonatomic, assign) CGSize shadowOffset
     Shadow offset. Optional property.
```

5.14 DTAliasRequest

The DTAliasRequest class is used to specify how aliases are obtained.

5.14.1 Class Methods

There are no class methods.

5.14.2 Instance Methods

```
-(id) initWithMerchantId:(NSString *)merchantId
      paymentMethods:(NSArray *)paymentMethods
```

Creates a new alias request for credit card selection by library.

Parameters

merchantId
    Datatrans merchant ID

paymentMethods
    Selectable credit card payment methods for alias generation.

```
-(id) initWithMerchantId:(NSString *)merchantId
           cardPaymentMethod:(DTCardPaymentMethod *)method
        verifyingTransaction:(BOOL)verify
```

Creates a new alias request with given credit card data.

Parameters

merchantId
    Datatrans merchant ID

cardPaymentMethod
    Credit card data

verifyingTransaction
    Whether to place a test authorization transaction of a small amount to verify the credit card.

5.15 DTSwissBillingPaymentInfo

Data container for SwissBilling transactions. A DTSwissBillingPaymentInfo object has to be set as payment option if SwissBilling is used or may be chosen by user.

5.15.1 Instance Methods

```
- (id) initWithCustomerAddress:(DTAddress *)customerAddress
                         phone:(NSString *)phone
                         email:(NSString *)email
                     birthDate:(DTDate *)birthDate
```

Creates a new DTSwissBillingPaymentInfo object.

Parameters

customerAddress
Customer address, mandatory

phone
Customer phone number, mandatory

email
Customer email, mandatory

birthDate
Customer birth date, mandatory

5.15.2 Properties

```
@property (nonatomic, assign) NSInteger taxAmount
     Total tax amount of order

@property (nonatomic, copy) NSString* customerId
     Customer ID

@property (nonatomic, copy) NSString* customerLanguage
     Language code for SwissBilling pages (overrides library language)

@property (nonatomic, copy) DTAddress* shippingAddress
     Shipping address

@property (nonatomic, copy) NSArray* basketItems
     List of basket positions of type DTBasketItem
```

5.16 DTAddress

Object representing a user's address or a shipping address.

5.16.1 Instance Methods

```
- (id) initWithFirstName:(NSString *)firstName
                lastName:(NSString *)lastName
                  street:(NSString *)street
                    city:(NSString *)city
                 zipCode:(NSString *)zipCode
             countryCode:(NSString *)countryCode
```

Creates a new address object.

Parameters

firstName
First name

lastName
Last name

street

    First street line (see properties)

city

    City

zipCode

    ZIP code

country code

    ISO country code

5.16.2 Properties

```
@property (nonatomic, copy) NSString* street2
      Second street line
```

5.17 DTDate

Object representing a date in Datatrans format.

5.17.1 Instance Methods

```
- (id)  initWithYear:(int)year month:(int)month day:(int)day
```

Creates a new date object from date components.

Parameters

year

    Year (e.g. 2015)

month

    Month, 1-based (e.g. 1 for January)

day

    Day, 1-based (e.g. 1 for first day of month)

```
- (id)  initWithString:(NSString *)dateString
```

Creates a new date object from formatted string.

Parameters

dateString

    Formatted Datatrans date string, either dd.MM.yyyy, or yyyy-MM-dd

5.18 DTBasketItem

Data container for SwissBilling transactions (see DTPaymentOptions).

5.18.1 Instance Methods

```
- (id)  initWithId:(NSString *)articleId
           name:(NSString *)name
    grossPrice:(NSInteger)grossPrice
      quantity:(NSInteger)quantity
```

Creates a new basket position object.

Parameters

articleId

    Article ID

name

    Article name

grossPrice
Article price

quantity
Quantity ordered

5.18.2 Properties

```
@property (nonatomic, copy) NSString* itemDescription
     Description of this item

@property (nonatomic, assign) float_t tax
     Tax rate of this item

@property (nonatomic, assign) NSInteger taxAmount
     Tax amount

@property (nonatomic, copy) NSString* type
     Item type, "goods" is default
```

5.19 DTApplePayConfig

Configuration object for Apple Pay transactions (see DTPaymentOptions).

5.19.1 Class Methods

```
+ (BOOL)hasApplePay
```

Whether Apple Pay is available on the device. It is not usually necessary to check this as the library automatically hides the Apple Pay payment method if unavailable.

5.19.2 Instance Methods

```
- (instancetype)initWithMerchantIdentifier:(NSString *)merchantIdentifier
                               countryCode:(NSString *)countryCode
```

Creates a new Apple Pay configuration object with Apple Pay country code.

Parameters

merchantIdentifier
Merchant identifier registered for Apple Pay

countryCode
Apple Pay country code

```
- (instancetype)initWithMerchantIdentifier:(NSString *)merchantIdentifier
```

Creates a new Apple Pay configuration object with CH (Switzerland) country code for payments in Switzerland.

Parameters

merchantIdentifier
Merchant identifier registered for Apple Pay

5.19.3 Properties

```
@property (nonatomic, readonly) PKPaymentRequest* request
     The request object for additional configuration of Apple Pay. See official
     Apple Pay documentation for more information.

@property (nonatomic, weak) id<DTApplePayDelegate> delegate
     Delegate object for callbacks during Apple Pay authorization.
```

5.20 DTApplePayDelegate

Delegate object for callbacks during the Apple Pay authorization process. Methods are taken from `PKPaymentAuthorizationViewControllerDelegate` (minus non-optional methods implemented by the library itself). For more information, see the documentation of `PKPaymentAuthorizationViewControllerDelegate`.

6 Library Integration

6.1 Package Contents

The library is distributed as a single .zip file with a directory structure as shown in Table 6-1.

Directory	Description
`/doc`	Contains this documentation.
`/include`	Contains header files of public library classes.
`/lib`	Contains the static library code.
`/resources`	Contains resources used by the library.

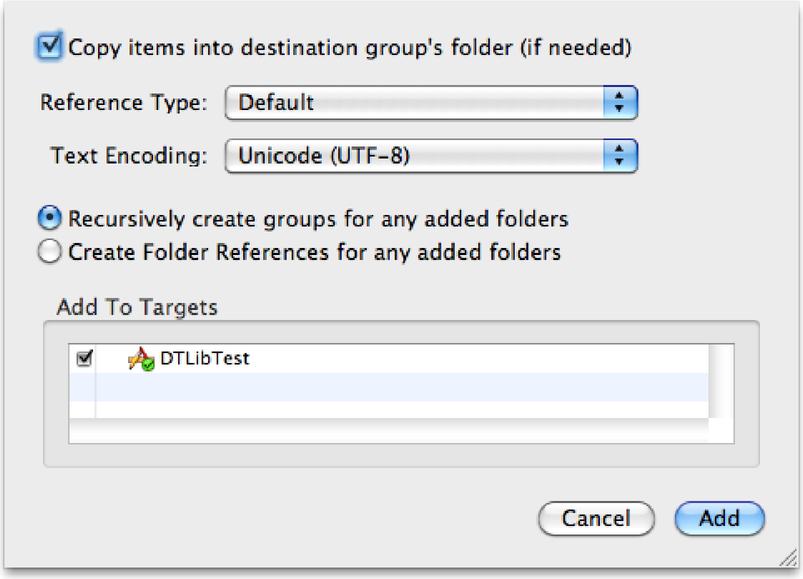
Table 6-1: Directory structure

In order to use the library in a new project, these files have to be copied into the project's Xcode environment.

6.2 Xcode Integration

Open your project file. Right-click on the project in Xcode and choose Add->New Group. Use DTiPL as the group's name.

Drag and drop the include and the lib folder of the library distribution into the newly created DTiPL group. Make sure "Copy items into destination group's folder" is checked and that items are added to your targets (Figure 6-1).



☑ Copy items into destination group's folder (if needed)

Reference Type: Default

Text Encoding: Unicode (UTF-8)

● Recursively create groups for any added folders

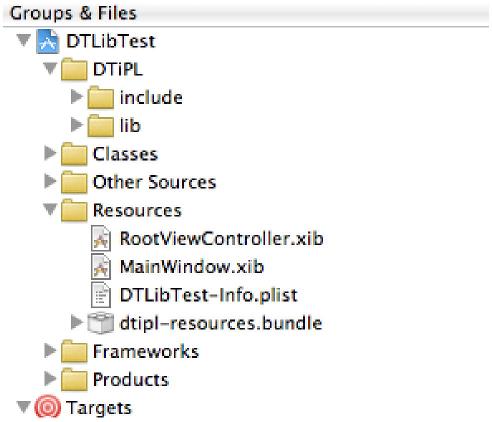
○ Create Folder References for any added folders

Add To Targets

☑ DTLibTest

Figure 6-1: Copying files into Xcode

Copy library file resources/dtipl-resources.bundle to the Resources folder of your project. The project should now look as depicted in Figure 6-2.



```
Groups & Files
▼ DTLibTest
    ▼ DTiPL
        ▶ include
        ▶ lib
    ▶ Classes
    ▶ Other Sources
    ▼ Resources
          RootViewController.xib
          MainWindow.xib
          DTLibTest-Info.plist
        ▶ dtipl-resources.bundle
    ▶ Frameworks
    ▶ Products
▼ Targets
```

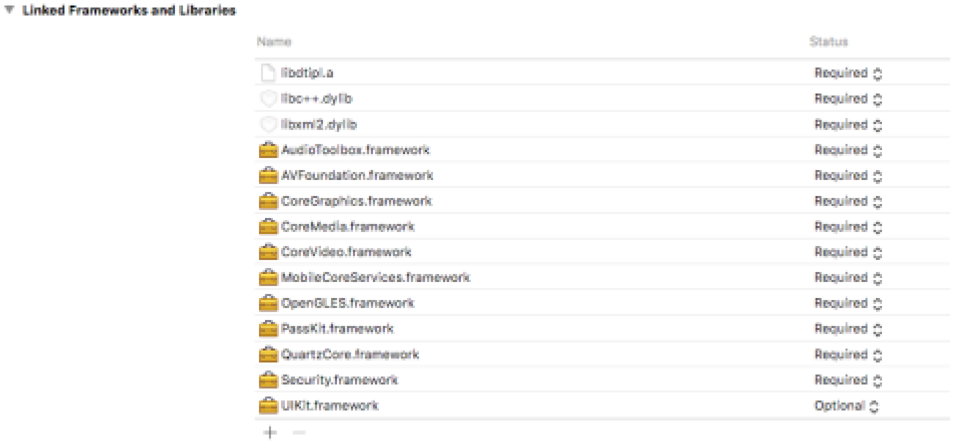
Figure 6-2: Project structure

Click the project file and **add `-all_load -ObjC`** to Other Linker Flags under the build tab for all configurations.

Click the app target. Under the general tab add the following libraries to the list of *Linked Frameworks and Libraries*:

- libdtipl.a
- libc++.dylib
- libxml2.dylib
- AudioToolbox.framework
- AVFoundation.framework
- CoreGraphics.framework,
- CoreMedia.framework
- CoreVideo.framework
- MobileCoreServices.framework
- OpenGLES.framework
- PassKit.framework
- QuartzCore.framework
- Security.framework
- UIKit.framework

If you are targeting iOS 5, please make sure to weak-link `UIKit.framework` by setting the framework's status flag to *optional*. A sample configuration can be seen in Figure 6-3.



Linked Frameworks and Libraries

Name	Status
libdtipl.a	Required
libc++.dylib	Required
libxml2.dylib	Required
AudioToolbox.framework	Required
AVFoundation.framework	Required
CoreGraphics.framework	Required
CoreMedia.framework	Required
CoreVideo.framework	Required
MobileCoreServices.framework	Required
OpenGLES.framework	Required
PassKit.framework	Required
QuartzCore.framework	Required
Security.framework	Required
UIKit.framework	Optional

Figure 6-3: Frameworks and libraries to be linked

The library’s header files can now be included into class files and the project builds and links with the Datatrans iOS payment library.

7 Known Issues

7.1 Bitcode

Starting from Xcode 7, apps are built with *Bitcode* by default. Bitcode is intermediate program code that can be compiled to a final binary differently for specific devices. Currently, Bitcode is rarely required and usually not useful. On the other hand, it makes intermediate products such as the Datatrans Payment Library significantly bigger. The library's size with Bitcode is about 170 MB.

If you are determined to use Bitcode in your app, please contact Datatrans Support for a version that has Bitcode enabled.

For everyone else, we suggest to turn off Bitcode in your project's settings:

Build Settings -> Build Options -> Enable Bitcode: NO

If you do not disable Bitcode, your project continues to compile and run fine. However, you will get an error once you are trying to archive the app for iTunes Connect:

```
ld: bitcode bundle could not be generated because 'lib/libdtipl.a(libdtipl.a-arm64-
                master.o)' was built without full bitcode. All object files and
                libraries for bitcode must be generated from Xcode Archive or
                Install build for architecture arm64

clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

8 Appendix

8.1 References

Number	Document	Version
1		
2		
3		

Table 8-1: List of references to other documents

8.2 List of Illustrations

8.3 List of Code Listings

8.4 List of Tables